LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

O VI. NÚMERO 55

UNA PUBLICACIÓN DE: TOWER COMMUNICATIONS

975 Ptas. • 5,86 € (IVA incluido)

IUEVAS TECNOLOGÍAS
ovedades y cambios con Java 2

AVA
ratamiento de imágenes

ITML DINÁMICO
reación de un banner

PROGRAMACIÓN GRÁFICA
fectos atmosféricos con una 3Dfx

NTRANET
el HTML al acceso a BBDD

BASES DE DATOS
cceso a Oracle 8 con C++ Builder

CREAR aplicaciones con VIDEOCONFERENCIA

8 413042 303299 00055

CONTENIDO DEL CD-ROM

DIRECTX 6.1 SDK • FILEMAKER PRO SERVER 3.0 • JPERK 5.0 • IBM VISUALAGE FOR JAVA ENTRY 2.0

SÓLO PROGRAMADORES

Número 55
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS
Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Subdirector
Oscar Rodríguez Fernández
oscarrf@towercom.es
Coordinador Técnico
Eduardo De Riquer Frutos
eriquer@towercom.es
Coordinadora de Redacción
Erika de la Riva Arnáiz
eriva@towercom.es
Colaboradores
Constantino Sánchez, Juan M. Menéndez, Jorge Delgado, Javier Sanz, Adolfo Aladro, Enrique de la Lastra, Rafael Corchuelo, Victoria Rus
Maquetación y Tratamiento de Imagen
Ana Isabel Madero Bocos

Publicidad
Inmaculada Romera (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Alicia Zazo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Javier Amado (Jefe)
jamado@towercom.es
Servicio Técnico
Manuel Hernando
mhernando@towercom.es

Preimpresión
IndesColor
Impresión
Gráficas Muriel
Distribución
SGEL
Distribución en Argentina / Chile / Colombia / México / Venezuela
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es
Distribución
Almiro Sanguino

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 30-6-99

Editorial

Parque informático jurásico

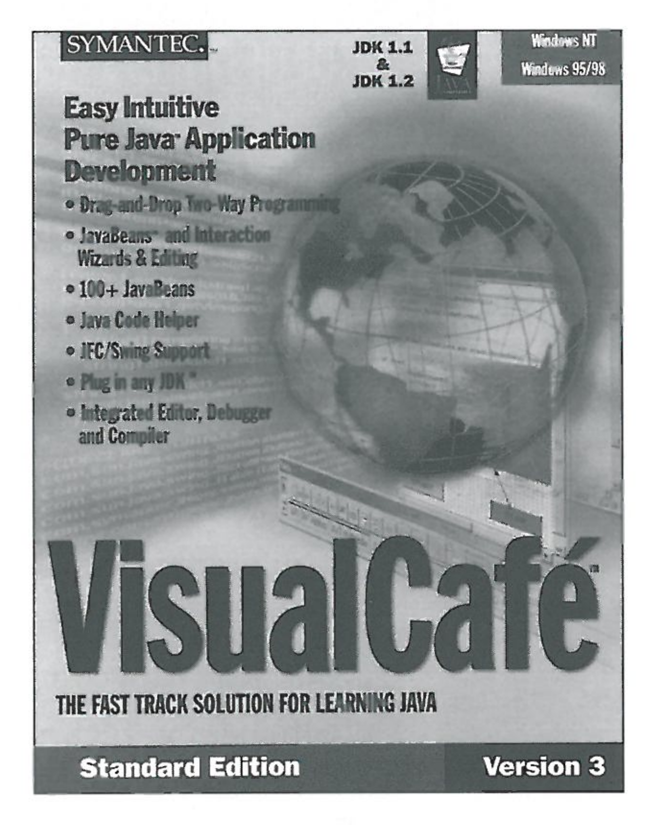
Aquí nos encontramos de nuevo, confundidos por los desplantes de nuestra querida tecnología, tan cercanos al próximo siglo que ya no resulta desbaratado pensar en la implantanción de un chip con funciones de eurocalculadora. Eso sí, que nos juren que no presentará problemas con el año 2000, que no requerirá parches ni *Service Packs* ni por supuesto realizar nuevas intervenciones. Tanta prevención, aunque razonable, podría parecer excesiva, aunque en vista del éxito obtenido por otros muy poderosos y aventajados lo mejor es que lo solicitemos por escrito. A veces resulta increíble que el futuro, signo de innovación y de avance continuo, tenga como consecuencia el retorno a los orígenes. No me refiero a nada que tenga que ver con profecías, ni siquiera con curvas o períodos cíclicos, ni al Renacimiento informático, sino únicamente la vuelta a los lenguajes que se utilizaban cuando los *Beatles* comenzaban su carrera. Que nadie se sorprenda si les digo que una buena forma de encontrar trabajo hoy en día "sólo" requiere un buen dominio de lenguajes como *COBOL* o *FORTRAN*, por ejemplo. Algunos, tras mucho discurrir obtendrán un aviso del tipo *"Key not found"*, "no computable" o algo así desde sus cerebros, pero estos lenguajes siguen formando parte muy activa en el mundo informático actual.

Este desconocimiento resulta normal cuando su apogeo universitario y académico de estos lenguajes procede de principio de los ochenta, lo que quiere decir que en plan de estudios no se incluye referencia alguna sobre este tema desde hace ya bastantes años. Además, cuando se impartían como asignaturas tampoco se prestaba excesiva atención, sobre todo desde el preciso momento en que empezaba a tener éxito un joven altanero llamado *C*, que evolucionó más y más y que acabó dominando el panorama con su orientación a objetos. De lo anterior se deduce que existen multitud de empresas que disponen de enormes ordenadores funcionando con lenguajes de este nivel, y cuando llega la oportunidad obligatoria de actualizarse la reacción no consiste en mirar hacia delante, sino en rescatar a los programadores pioneros para parchear el problema. Y me pregunto, ¿cuándo tienen previsto efectuar el cambio, si es que lo tienen? ¿qué ocurrirá cuando no sea posible recurrir a programadores especializados en estas áreas más que olvidadas? ¿pasará la solución por el uso de la tradición oral? Por favor, que nadie se ría, porque debemos ser agradecidos con la plataforma sobre la que evolucionaron los lenguajes de mayor actualidad. Además, no resulta tan simple permanecer operativo durante más de cuarenta años (otro siglo), sobre todo teniendo en cuenta las enormes diferencias que existían entre las épocas comparadas. Ya veremos dentro de cuarenta años dónde se encuentran los triunfadores de hoy (acompañados o no de ventanas, que es otro tema) y cuánto y de qué forma soportarán a los impulsivos hijos que puedan tener. En cualquier caso, nunca vendría mal un poco de planificación y sobre todo la posibilidad de acogerse a una buena jubilación anticipada, a ser posible muy anticipada.

SÓLO PROGRAMADORES



PRESENTACIÓN DE VISUAL CAFÉ 3.0



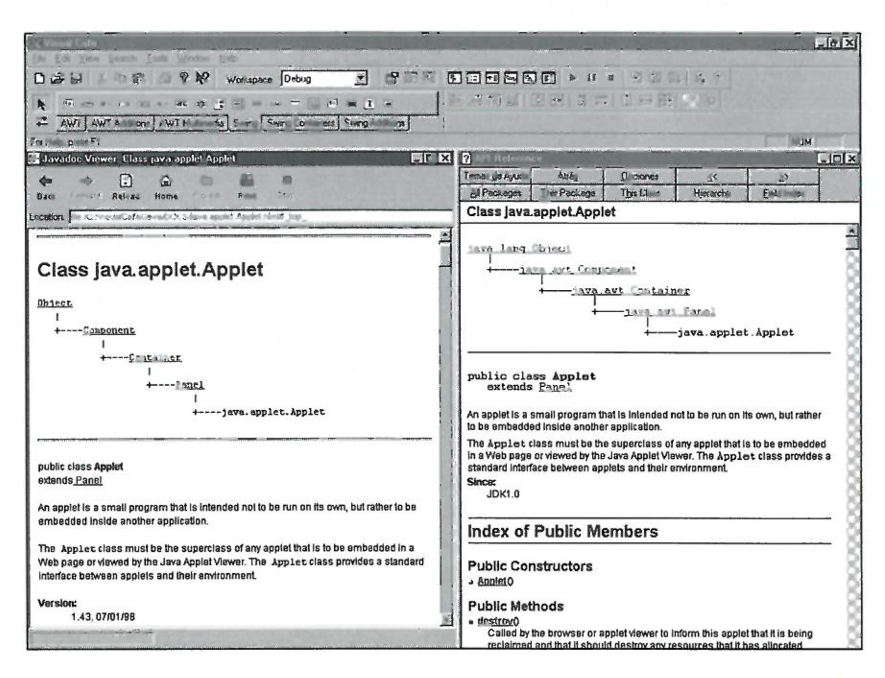
Con el lanzamiento de *Visual Café 3.0*, *Symantec* ha dado un importante paso hacia delante en la construcción de entornos *RAD* para Java. El desarrollador encontrará todo un entorno adecuado a sus requerimientos y necesidades mediante el uso de cualquiera de los tres entornos disponibles en esta versión, *Standard*, *Profesional* y *DataBase*.

Se han mejorado notablemente las características del producto sobre versiones anteriores y la incorporación de nuevas herramientas y de un conjunto de nuevos asistentes hace que con Visual Café 3.0 el desarrollo de aplicaciones sea una tarea una aún más fácil y sencilla si cabe.

Muchas de las posibilidades existentes han sido mejoradas. Se han corregido ciertos problemas que había con la recuperación de los proyectos, añadido nuevas opciones en los menús del entorno para controlar las opciones de compilación, con lo que ya no es necesario modificar el fichero *sc.ini* manualmente y el conjunto de componentes ha sido actualizado y ampliado. Además, igualmente sorprende el incremento de velocidad, tanto del entorno como de las aplicaciones que se generan.

Entre las nuevas características destacan el sistema *two-way drag and drop*, que permite el desarrollo simultáneo tanto visual como escrito, observándose así el resultado en tiempo real, *Code Helper*, que ayuda a corregir "en línea" problemas en el código mediante la indicación de los parámetros de una función, tipos erróneos, etc., y todo un conjunto de nuevas herramientas para el desarrollo con bases de datos.

En el próximo número de la revista encontrara un artículo de evaluación donde se describirá Visual Café 3.0 detalladamente.

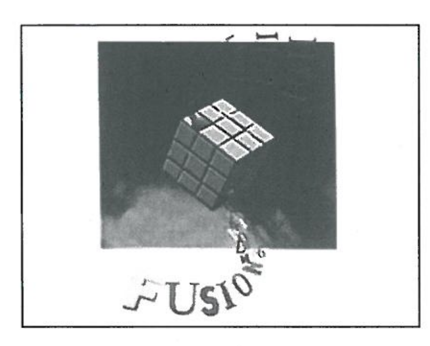


INFORMATION BUILDERS ANUNCIA FOCUS FUSION, LA PRIMERA BASE DE DATOS MULTIDIMENSIONAL

Recientemente ha sido presentada por la compañía Information Builders FOCUS Fusion para entornos MVS. Se trata de la primera base de datos multidimensional y de alto rendimiento disponible para plataformas MVS. Las capacidades de particionamiento e indexación de ésta permiten acceso directo y de alta velocidad a los datos, incluso cuando se consultan detalladamente y en cantidad.

Estas características ofrecen a los usuarios una enorme flexibilidad a la hora de realizar un análisis. El particionamiento inteligente también permite planificar actualizaciones, mientras el resto de la base de datos está disponible para llevar a cabo procesos de análisis.

FOCUS Fusion se complementa con un potente conjunto de herramientas que forman parte de la familia SmartMart Data Warehousing Suite, y también hace uso del potente middelware EDA (Enterprise Data Access). Para ampliar la información



sobre esta base de datos puede consultar en la siguiente dirección: www.ibi.com

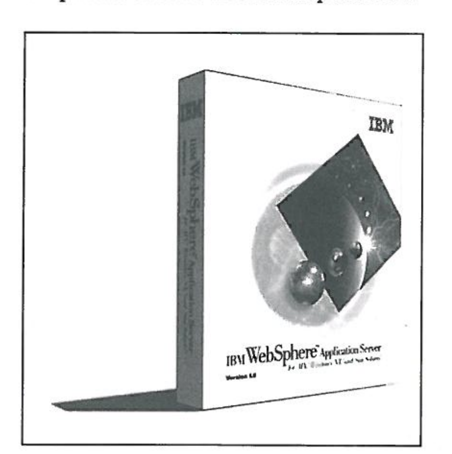
IVES DEVELOPMENT LANZA TEAMSTUDIO LIBRARIAN

Acaba de salir al mercado TeamStudio Librarian, la última incorporación a la gama TeamStudio de herramientas de desarrollo para Lotus Notes y Domino. Este producto permitirá el principio de reutilización de código en un equipo de desarrollo Notes & domino, al acceder a que los promotores desarrollen archivos de elementos de diseño compatibles, como subformatos, archivos de texto y archivos de agentes.

Esta herramienta estará disponible para los usuarios a partir del día uno de abril de este año y el precio establecido es de 425 dólares por asiento de promotor e incluye apoyo técnico ilimitado y mantenimiento durante un año. Si quiere ampliar la información puede visitar la página web www.teamstudio.com

IBM INTRODUCE NUEVAS FUNCIONALIDADES EN WEBSPHERE APPLICATION SERVER

IBM ha introducido una serie de mejoras relacionadas con el rendimiento, la seguridad y el soporte de plataformas en IBM WebSphere Server, solución que proporciona a las empresas todo lo necesario para crear y gestionar un servidor web basado en estándares.



Todas estas mejoras están destinadas a ofrecer a los administradores y desarrolladores web la fiabilidad, seguridad y rendimiento que proporciona habitualmente IBM en sus soluciones empresariales.

Incluidas entre las novedades se encuentra por ejemplo la introducción de una nueva tecnología de IBM Research, denominada FRCA que incrementa en un 100% el rendimiento de los servicios HTTP. Otra interesante mejora es el soporte de SSL, que proporciona a los clientes la seguridad necesaria para crear un sitio web preparado para operar con información confidencial como la que se genera habitualmente en las transacciones financieras.

WebSphere Application Server es la base de esta familia de aplicaciones y proporciona a las empresas una plataforma abierta y basada en estándares para crear servidores web, así como también el conjunto de herramientas necesarias para gestionar un sitio web.

El servidor HTTP Apache se suministra junto con WebSphere Application Server. Puede obtener mayor información sobre este producto en: www.software.ibm.com/websphere

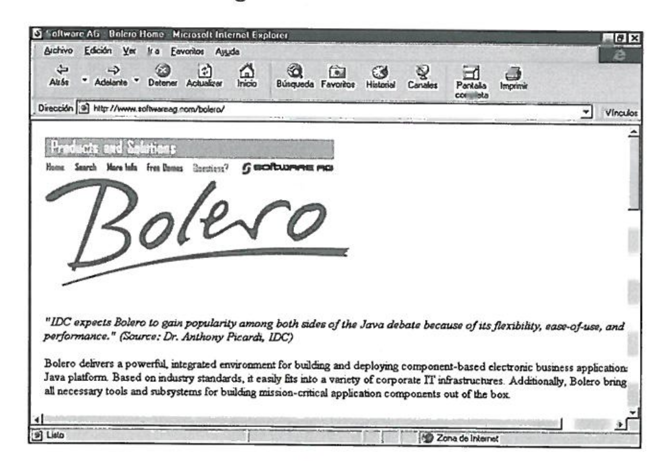
SOFTWARE AG ANUNCIA EL SOPORTE CICS EN BOLERO

Software AG acaba de anunciar que su herramienta de desarrollo de aplicaciones sobre plataforma Java, Bolero, soportará el servidor de transacciones CICS. Con esto se permitirá a los programadores crear componentes de software de misión crítica para aplicaciones web con un flujo intenso de transacciones en un entorno bastante moderno de programación.

IBM y Software AG han colaborado durante largo tiempo para conseguir que su robustez tecnológica sea una gran ventaja para sus clientes. Como resultado de ello ambas compañías han anunciado el soporte a la tecnología de componentes Enterprise Java Beans. Esto permitirá el uso de componentes desarrollados con Bolero en las nuevas versiones del servidor de transacciones CICS de IBM para la plataforma OS/390.

Bolero es el nuevo entorno de desarrollo e integración de aplicaciones orientado a objetos. Permite el desarrollo de componentes de software de misión crítica para entornos transaccionales sobre plataforma Java.

Existe más información a su disposición en la página web www.softwareag.com



JINI, EL NUEVO LENGUAJE DE SUN

Sun Microsystems presenta su nuevo lenguaje JINI. Éste será capaz de interconectar todo tipo de aparatos entre sí o también con un ordenador. Pretende ser un estándar que comunique impresoras, móviles, escáneres o incluso electrodomésticos.

Con su tecnología este lenguaje permite a cualquier dispositivo conectarse a la red e informar de su disponibilidad o capacidad, de este modo conectar un periférico puede llegar a ser tan sencillo como pulsar un botón, sin necesidad de configuración.

Este novedoso lenguaje sale al mercado con el apoyo de una treintena de grandes empresas del sector, que han licenciado la tecnología. Además JINI será probado por Symbian, la alianza de los tres grandes del sector móviles, (Nokia, Ericsson y Motorola), con el fabricante de agendas electrónicas Psion. Esta alianza actualmente está fabricando un sistema alternativo al Windows CE de Microsoft, para las agendas electrónicas y que sirva como unión entre los móviles y éstas.

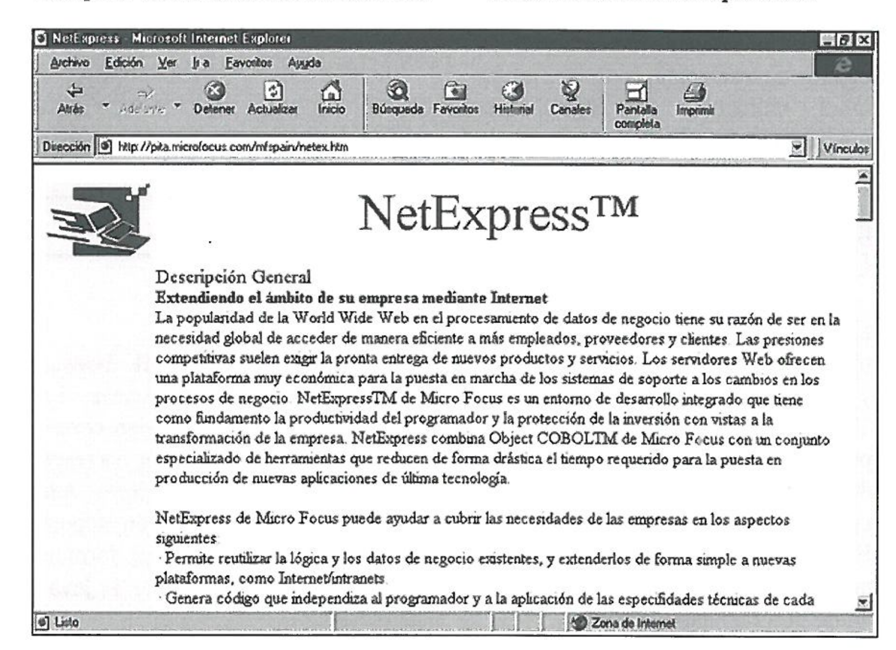
La dirección a la que puede acceder si desea obtener más información es www.sun.es

NETEXPRESS 3.0 DE MICROFOCUS LLEVA LAS APLICACIONES COBOL TRADICIONALES A INTERNET

MicroFocus ha comunicado el lanzamiento de NetExpress 3.0, un revolucionario entorno de desarrollo para aplicaciones e-COBOL, gracias al cual las empresas podrán trasladar sus programas escritos en COBOL a la Web y a otras plataformas de informática distribuida.

Además, puede utilizarse para la migración de aplicaciones a plataformas Windows NT y UNIX, así como para el desarrollo de nuevas aplicaciones distribuidas que utilicen componentes corporativos escritos en COBOL. Con todo esto la compañía ofrece a las grandes empresas la oportunidad de llegar a nuevos usuarios de una forma muy rápida, eficaz y sobre todo rentable.

NetExpress 3.0 ya se encuentra disponible y puede adquirirse llamando directamente al número de teléfono 902 210 103 o bien a través de cualquier distribuidor autorizado. Del mismo modo si desea obtener más información consulte la página web www.microfocus.com/spain.htm



HERRAMIENTAS DE PROGRAMACIÓN

ENTORNOS DE DESARROLLO

MICROSOFT DIRECTX 6.1 SDK

Kit de desarrollo de la ultimísima versión de DirectX. Paquete que Microsoft pone a disposición de los desarrolladores para que estos implementen tecnología DirectX a sus aplicaciones. Especialmente indicado para la programación de aplicaciones multimedia y juegos en que se haga uso intensivo de los gráficos y el sonido.



DRUMBEAT 2000

Excelente entorno para la creación de sites Web con contenido dinámico de bases de datos. Permite trabajar con Microsoft Internet Information Server y Personal Web Server. Su utilización, dentro de su complejidad, es sencilla. Pueden crearse páginas con efectos multimedia, filtros DHTML, efectos de transición, controles multimedi, validación de formularios, modificaciones de los CSS, y de muchos otros modos.



BUTTONWIZ (PROGRAMMERS EDITION) 5.1

Entorno que permite la creación simple de botones para cualquier interfaz y lenguaje. Están disponibles más de 150 tipos diferentes de botones que pueden ser modificados y personalizados. El trabajo final puede ser salvado en alguno de estos formatos PNG, BMP, TIF, JPG, PCX ó TGA. Además pueden ser llevados a cualquier entorno de programación gráfico.

CORELWORDPERFECT 8.0 PARA LINUX

La herramienta ofimática imprescindible del mundo Linux. Este es el producto resultado del primer acercamiento seriopor parte de la empresa Corel al mundo Linux. Poco se puede decir de este paquete tan conocido con la única salvedad de que en su versión Linux no ha perdido ni un ápice de potencia con respecto a las versiones Windows.

ALICE

Herramienta de autor freeware para la creación de mundos 3D para un entorno Web. Es un entorno primario de scripting para el uso de objetos 3D, no es un modelador. A través de simples scripts, Alice puede tomar el control de la apariencia de los objetos y de cómo y cuando deben ser visualizados en la Web. También soporta ficheros DXF y OBJ.

LENGUAJES

HTML



ARACHNOPHILIA 3.9

Completísimo editor HTML freeware de excelentes características. Las páginas desarrolladas pueden verse a través de previews internas o a través de seis diferentes navegadores. Además de todo esto también soporta tags de HTML 4.0, frames, formularios, marcos, CGI, Perl, C++, Java y Javascript.

COOLBE 1.2

Creación sencilla de banners animados para páginas Web. Se puede copiar, editar o modificar el diseño de los textos y los gráficos de los ejemplos predefinidos para usar en las páginas Web propias. También las dimensiones, los colores y la velocidad y el estilo de la animación. Es freeware.

IMAGECRUSHER 1.04

Creación de mapas de imágenes para la navegación Web. Las imágenes para la creación de los mapas pueden ser llamadas directamente del fichero HTML y al mismo tiempo creadas las áreas para facilitar la navegación.

LINKBOT PRO 4.0

Se trata de una interesante utilidad para el chequeo de errores en site Web. Permite realizar tests en los links internos y externos para a continuación crear una página HTML con la estructura del Web site y los resultados de los links erróneos.

JAVA

J-PERK 5.0

Creación rápida y sencilla de animaciones basadas en Java. Es ideal para la creación de slide shows, gráficos animados, cabeceras, botones dinámicos, scrolls de texto, imágenes en rotación. Basta con elegir la imagen y a continuación seleccionar las propiedades del movimiento. Con todo esto Java Perk crea el código HTML quea continuación debe insertarse en el documento Web.



APPLET PASSWORD WIZARD 1.1

Asistente para la creación de applets Java con password. Permite crear ilimitado número de entradas con un número de usuario y password. Trabaja en un frame o una tabla y permite controlar el estilo de los bordes, el tamaño y el color de fondo del applet.

VISUALAGE FOR JAVA ENTRY 2.0

Herramienta de desarrollo de aplicaciones Java rápida y potente. Con VisualAge se puede añadir una clase, incorporar o modificar un método y después compilar sin tener que salir del entorno de test. También incluye Visual Composition Editor, un entorno integrado basado en repositorio que proporciona control completo de la versión y de las fuentes. *Nota: IBM no se hace responsable de las posibles consultas técnicas.*

VISUAL BASIC

PROJECT ANALYZER 5.0 (VBSHOP) 5.0.05

Analizador de código fuente y generador de documentación Project Analyzer puede detectar código erróneo de Visual Basic, de forma que permite aumentar el potencial del programa desarrollado y disminuir su tamaño eliminado entradas innecesarias.

CGI WORKERMAN 1.5

Pequeño módulo de creación de script CGI desde Visual Basic. Proporciona la decodificación de las URL y pequeñas funciones de procesamiento de datos. Está especialmente indicado para su uso con servidores Web que no soporten CGI.

OTROS

ASP-EDIT 1.012B BETA

Editor para la creación de Active Server Pages. ASP-EDIT ofrece menús para la inserción de funciones, variables, VBScripts, etc. Incluye asistentes para la creación de estamentos SQL y para la inclusión de ficheros GLOBAL.ASA.

B-FLAT 0.30

Compilador de C que implementa ANSI standard C. B-Flat compila el código fuente para la creación de un programa compatible con ensamblador. El resultado es muy similar a un programa MS-DOS COM.

XML SPY 1.3

Es un editor XML y DTD. Se trata de una solución interesante para la creación y visualización de código XML y DTD. Incluye visualización de datos estructurada, opciones varias de impresión, deshacer ilimitado, capacidades de buscar y reemplazar, etc.

HERRAMIENTAS /UTILIDADES

CD AUTORUN 1.0.1

Herramienta freeware para la creación de CD's con autoarranque. El programa permite personalizar completamente las pantalla de autoarranque de un CD-ROM. De esta manera es posible elegir las rutas de los ficheros, las imágenes que aparecerán, la inclusión de sonidos o las acciones de botones y textos resaltados.

BACKUP IT/32 2.01

Librería para añadir capacidades de backup a los desarrollos. Permiten al usuario de la aplicación desarrollada el hacer copias de seguridad y restaurarlas. Incluye características de compresión de archivos, gestión de atributos, inclusión de fechas, copia exacta de directorios, etc.

BUGFIX 2.0

Test completo de software y hardware para el año 2000. Utilidad freeware que realiza más de 40 chequeos distintos para identificar si la máquina es compatible año 2000. También permite imprimir los resultados para su posterior consulta.

MICROSOFT DIRECTX 6.1

Librerías para la ejecución de aplicaciones multimedia y juegos en Windows. Última actualización de la librería multimedia de Microsoft. Incluye soporte para la API Direct3D para los Intel Pentium III. Como novedad fundamental aporta DirectMusic que solventa la histórica limitación de los sistemas MIDI.

CYGWIN 1.0

Colección de herramientas para Windows con similares características que las GNU. Se trata de unas herramientas freeware que posibilitan el desarrollo bajo Windows del mismo modo que se realiza también bajo el API de Unix.

NEXENCODE 2.0

Encoder, decoder y CD ripper de ficheros MP3. Ofrece la codificación en un sólo paso, un interfaz personalizable, soporte para plug-ins, editor de composiciones y un editor de skins y formas. Incluye un decoder de MP3 a WAV. Es freeware.

PROGRAMMER'S FILE EDITOR (PFE) 1.01

Editor de textos freeware diseñado especialmente para desarrolladores. Entre otras características, se encuentra la de ofrecer la posibilidad de ejecutar comandos DOS y capturar su salida, plantillas, mapa de caracteres, macros, múltiples niveles de deshacer, algún soporte para archivos Unix, etc.

REDES

DISTRIBUIDAS

COOL INFO 99 2.5

Información de red del sistema para administradores. Cool Info muestra información sobre las unidades del sistema, ficheros, directorios, RAM, CPU, impresoras, red, fuentes, teclado, ratón, monitor, adaptadores gráficos, etc. Los datos pueden ser exportados a una base de datos o a un fichero de texto. Es freeware.

IMAIL SERVER 5.0

Servidor de correo electrónico de interesante características. Es muy sencillo de administrar y puede ser usado con los estándares de correo POP3 e IMAP4. Incluye características de protección anti-spam, soporte para cientos de usuarios, administración remota, conexión a bases de datos ODBC, soporte LDAP 3.0, etc.

SAM SPADE 1.10 BETA

Conjunto de 16 utilidades de red para Internet. Dispone de interfaz gráfico propio que integra todos los programas. Incluye herramientas como Ping, Whois, Finger, NSLookup y un traceador gráfico. Todo un kit para administradores de red que permite el control total de ésta.

LOCALES

CYBERGAUGE 2.0

Monitorización de procesos Internet a través de LAN o WAN. Permite controlar el acceso a Internet para disponer de los datos necesarios que a veces tan sólo tiene el ISP. CyberGauge usa SNMP (Simple Network Management Protocol) para monitorizar el paso de los datos que se envían y provienen de Internet.

KEYVISION 3.0

Herramienta de mantenimiento centralizado a través de la monitorización. A través de la información que reporta permite realizar una sencillo y potente actualización y mantenimiento de los equipos conectados a una red. Todo ello se realiza a través del Registro de Windows y los datos que se ocultan en él.

NETCONNECT 1.1

Creación de mapas de redes para su gestión gráfica. NetConnect simplifica la conexión, desconexión y mantenimiento de unidades remotas de red. A través de mapas con la letra de las unidades permite conectar y desconectar una red local a Internet. Del mismo modo permite gestionar un número ilimitado de unidades.

OTROS

FILEMAKER PRO SERVER 3.0

Servidor de bases de datos para empresas. Solución ideal para los administradores de red. Compatible con las bases de datos FileMaker Pro 3, 4, y 4.1, este motor de base de datos relacional de altas prestaciones acelera las operaciones FileMaker Pro en red de forma muy notable.

Además, permite a 100 usuarios consultar simultáneamente hasta 100 bases de datos.

NORTON GHOST 1.10

Creación de imágenes de discos duros. La imagen creada, con todos sus programas y configuraciones, puede ser simultáneamente copiada en varios PCs desarrollando instalaciones exactas en todos ellos.

Ofrece interfaz gráfico o batch mode, multicasting, ajuste automático del tamaño de la partición, compresión de la imagen, chequeo CRC, protección con password, comandos FDISK y FORMAT mejorados, etc.

OMNI-NFS ENTERPRISE 4.0

Solución de red empresarial para acceso a Unix. Ofrece características para el acceso a través de PCs Windows 95/98/NT a ficheros e impresoras de sistemas Unix. Se integra perfectamente en el entorno Windows y dispone de varias herramientas de conectividad.

SIMPLE DNS 1.03.02

Simple conversión de DNS por nombres de texto. Programa que permite nombrar con texto las DNS de una red de área local. Con Simple DNS el servidor E-mail puede llamarse "mail", el servidor proxy "proxy" y el servidor Web "server".

DOCUMENTACION /TUTORIALES

BEGINNING VISUAL BASIC 1.0

Tutorial interactivo para el aprendizaje de Visual Basic. El curso está diseñado para los más principiantes, no es preciso disponer de fundamentos de programación. Se basa en la creación de una aplicación explicando paso a paso como se realiza.

Se incluyen cinco lecciones que sondenominadas: "Introducing Visual Basic", "The Visual Basic Environment", "Your First Visual Basic Project", "Project Design, Forms, Command Buttons", y "Labels, Text Boxes, Variables".

BEGINNER'S GUIDE TO DHTML 1.0

Guía freeware para el aprendizaje de HTML Dinámico para principiantes. El

temario está basado en seis capítulos: "What is DHTML?", "DHTML in Netscape 4", "DHTML in Internet Explorer 4", "Dynamic Content", "Moving Elements Around In The Document", y "Creating Cross-Browser DHTML".

Se encuentra en formato HTML y puede ser visualizada en cualquiera de los navegadores conocidos.

REGEDIT.COM 1.0

Recopilación técnica sobre el Registro de Windows 95/98.Incluye trucos de los más avanzados para conseguir el máximo rendimiento de un PC con Windows. El tutorial, en formato .HLP, ayuda al usuario a realizar cambios en el Registro sin miedo alguno a la pérdida de datos.

Los temas son: performance and functionality enhancements, security restrictions and policies, hidden and useful windows functions, troubleshooting, y muchos más.

WEB RESOURCES' TUTORIALS 1.0

Pequeños tutoriales que incluyen texto técnico de aprendizaje sobre Javascript, DHTML, CSS, etc.. Incluye información de referencia sobre Javascript, cookies HTML, style sheets,HTML Dinámico (DHTML) y todo lo que tiene que ver con la programación para la Web.

Es freeware y se trata de una buena opción para todos aquellos usuarios que disponen de conocimientos básicos de HTML

HERETIC AND HEXEN SOURCE CODE

Código fuente completo en C de los juegos Heretic y Hexen de Raven Software. Activision y Raven se encuentran actualizando este código para que cualquier programador pueda aprender y jugar con él.

Este código no puede ser utilizadp para aprovecharse de él comercialmente. Además las librerías de sonido DMX originales no están incluidas con el código.

FUENTES DE LOS ARTÍCULOS

En esta interesante sección del CD-ROM os entregamos todos los códigos fuentes que los autores de los artículos nos facilitan, para que os resulte más sencillo, cómodo y agradable poder realizar las practicas, o seguir los ejemplos que han sido planteados en cualquiera de los artículos de las páginas de esta revista.

HTML Dinámico (V)
Ejemplos de diferentes efectos realizados con HTML Dinámico ./program/DHTML5/

Creación de un buscador Web (IV)
Interesante ejemplo del artículo planteado en la revista ./program/BUSCA4/

Windows Scripting Host (IV)
Los fuentes de la práctica propuesta en el artículo ./program/WSH4/

Intranet (I)
Los fuentes de la práctica propuesta en el artículo ./program/INTRANET1/

Tratamiento de las imágenes con Java (II)
Ejemplo práctico creado siguiendo los pasos que se explican en el artículo de este mes ./program/JAVA2/

Desarrollo de aplicaciones para videoconferencia
Incluimos el código fuente del ejmplo realizado en el artículo de este mes ./program/VIDEO1/

IMPRESCINDIBLES

ANTIVIRUS

McAfee VirusScan
Panda Antivirus Platinum

GRAFICOS

The Best Icons
LView Pro Image Processor 2.1
Paint Shop Pro 5.01
ThumbsPlus 3.30s
Xara 3D 3

INTERNET

AutoWinNet 5.5 Beta 2
CuentaPasos 3.51
CuteFTP 2.6
Eudora Light 3.0.6
Go!Zilla 3.3
HomeSite 4.0
mIRC 5.5
URL Organizer 2.0
WebZIP 2.50

MULTIMEDIA

GoldWave 4.02
WinAmp 2.05

NAVEGADORES

Netscape 4.08
Opera 3.51

RUNTIMES

Runtimes de Visual Basic

UTILS

Adobe Acrobat Reader 3.01
Babylon Translator
CDR Win
Day Time Organizer 2000
DirectX 6.0
Fix 2000
Cybermedia Uninstaller
Windows Commander 3.53
Win32s
WinZip 7.0



HTML Dinámico (y V). Crear un banner

Adolfo Aladro (aaladro@arrakis.es)

En el mundo de Internet un *banner* (estandarte, bandera, pancarta) no es más que un reclamo, publicitario o no, que se coloca en una parte destacada de la página *HTML* y que sirve para llamar la atención del visitante y motivarle para que haga clic sobre él, con lo que saltará al sitio *Web* que se trata de promocionar.

Al principio se trataban de imágenes estáticas, aunque con el tiempo se fueron incorporando las imágenes GIF animadas, que hoy día copan la mayor parte de los *banners*. Prácticamente cualquier cosa es válida para crear un *banner* que capte el interés del que lo ve. En este sentido el *HTML* Dinámico ofrece un método sencillo y ágil de crear estas pancartas poniendo a nuestra disposición un amplio abanico de posibilidades. A lo largo de esta serie de artículos vamos a ver cómo podemos hacer distintos *banners*.

ASPECTO DE UN BANNER

El aspecto que presentan todos los *banners* suele ser el mismo. Un rectángulo que ocupa normalmente la zona superior de la pantalla (No en vano las estadísticas han demostrado que las partes de las páginas *Web* que se visualizan moviendo la barra de *scrolling* son las menos vistas). Dentro de ese rectángulo se encuentran imágenes o texto que pueden cambiar cada cierto tiempo y además pueden ser al mismo tiempo enlaces. A veces también nos encontramos con alguna caja de texto donde podemos introducir, por ejemplo, nuestra dirección de correo electrónico, o bien una lista desplegable donde seleccionamos alguna opción. Hasta aquí no estamos hablando de nada que no podamos conseguir con *HTML* Dinámico con un poco de habilidad.

Las ventajas que ofrece la realización de *banners* de este tipo utilizando *HTML Dinámico* son evidentes. Mientras un *GIF* animado es algo estático, a pesar de su animación, un *banner* basado en *HTML* Dinámico puede variar o modificarse en cualquier momento. Además es capaz de adaptarse a las nuevas circunstancias de forma sencilla (Para cambiar el contenido de un *GIF* animado habría que volver a editarlo mientras que si se trata de un *banner* basado en *HTML* Dinámico normalmente sólo habrá que modificar una simple cadena de texto).

UN BANNER, UNA CAPA

Evidentemente desde el punto de vista del *HTML* Dinámico, un *banner* va a ser una capa que posicionaremos en la página *HTML* en el lugar que deseemos. Modificaremos sus parámetros y aspecto accediendo a sus propiedades de estilo e

incluso cambiando su propio contenido cuando así lo consideremos oportuno. Teniendo en cuenta estas premisas, el diseño de nuestro *banner* personal puede ser tan complejo como deseemos. En nuestro caso particular vamos a partir de un ejemplo sencillo que siente las bases a partir de las cuales el lector pueda sentirse libre para crear.

```
...
...
<BODY onload="iniciar()">
<DIV ID="banner" CLASS="bannerestilo">
<SCRIPT LANGUAGE="JavaScript">
<!—
 document.write(banner_codHTML(1));
//—>
</SCRIPT>
</DIV>
</BODY>
```

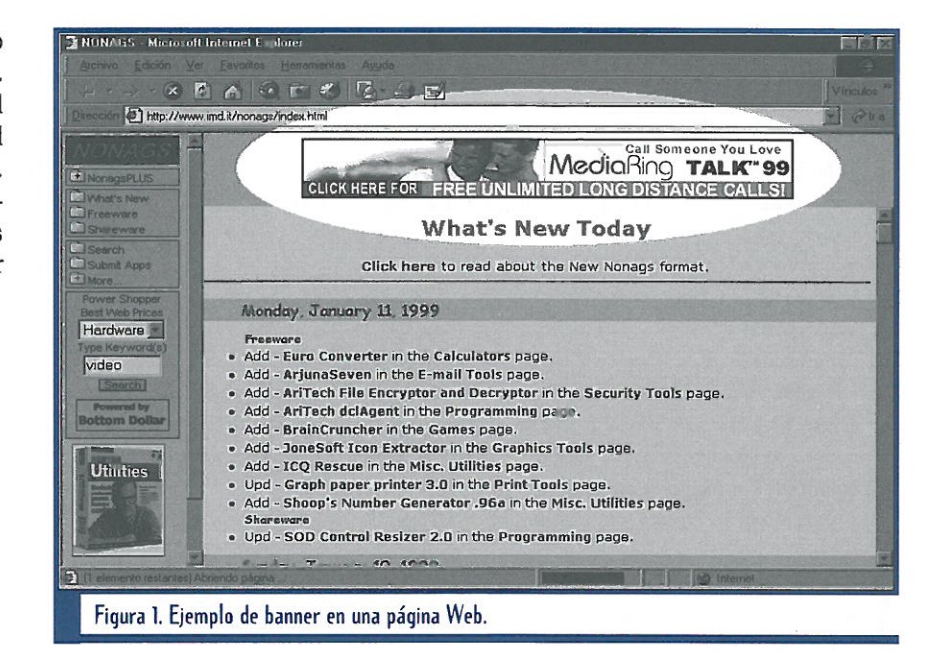


Figura 1. Ejemplo de banner en una página Web.

El código *HTML* correspondiente al contenido del *banner* se escribe de forma dinámica, de tal forma que para insertar el *banner* en nuestra página *Web*. Como vemos esto se hace dinámicamente mediante *Javascript* para darnos la oportunidad de generar distintos parámetros en el mismo momento que la página se carga. Por ejemplo, si queremos centrar el *banner* en la página necesitamos saber cuál es el tamaño de la ventana del navegador. Además de esto, las características del *banner* se han parametrizado mediante una serie de variables *Javascript* que describimos al principio del *script* de la página. Estas variables se utilizarán para configurar el aspecto del *banner*, en otras palabras, sería para generar el estilo correspondiente de la capa así como el código *HTML* que devuelve la función *banner_codHTML(estado)*.

```
<SCRIPT LANGUAGE="JavaScript">
<!—

banner_ancho = 400;
banner_alto = 50;
banner_alineacion = 'CENTER';
banner_posY = 10;
banner_colorfondo = '#EEEEEE';
banner_letra_fuente = 'Verdana';
banner_letra_tamanyo = '14pt';
banner_letra_peso = 'bold';
banner_letra_estilo = 'normal';
banner_letra_color = '#2020FF';
banner_texto_alineacion_vertical = 'MIDDLE';
```

Listado 1. Función que se encarga de generar dinámicamente el estilo de la capa que contiene el banner.

```
function estilos() {
codHTML = '';
codHTML += '<STYLE TYPE="text/css">';
codHTML += '.bannerestilo {';
codHTML += 'position: absolute;';
codHTML += 'visibility: hidden;';
codHTML += 'font-family: ' + banner_letra_fuente + ';';
codHTML += 'font-size: ' + banner_letra_tamanyo + ';';
codHTML += 'font-weight: ' + banner_letra_peso + ';';
codHTML += 'font-style: ' + banner_letra_estilo + ';';
codHTML += 'color: ' + banner_letra_color + ';';
if (NS4) {
 if (banner_alineacion == 'CENTER') {
  codHTML += 'left: ' + ((window.innerWidth - banner_ancho)/2) + 'px;';
 } else if (banner_alineacion == 'LEFT') {
  codHTML += 'left: 0px;';
 } else if (banner_alineacion == 'RIGHT') {
  codHTML += 'left: ' + (window.innerWidth - banner_ancho) + 'px;';
 }
}
codHTML += 'width: ' + banner_ancho + 'px;';
codHTML += '}';

codHTML += '</STYLE>';
document.write(codHTML);
}
```

Listado 2. Función que generará el código HTML correspondiente al contenido del banner.

```
function banner_codHTML(estado) {
 codHTML = '';
 codHTML += '<TABLE BORDER="' + banner_borde_tamanyo + '" WIDTH="' +
(banner_ancho-1) + '" BGCOLOR="' + banner_colorfondo + '" HEIGHT="' + (ban-
ner_alto-1) + '" CELLSPACING="0" CELLPADDING="' +
banner_espacio_borde_contenido + '"><TR><TD VALIGN="' + banner_texto_alinea-
cion_vertical + '">';
 if (estado) {
  codHTML += '<TABLE WIDTH="100%" HEIGHT="' + (banner_alto-1) + '" BOR-
DER="0" CELLSPACING="0"><TR><TD WIDTH="50%" BGCOLOR="' +
banner_colorfondo_izda + '" ALIGN="' + banner_alineacion_hor_izda + '"
VALIGN="' + banner_alineacion_ver_izda + '">';
  codHTML += '<IMG SRC="dados.gif" WIDTH=111 HEIGHT=48 BORDER=0
ALT="">';
  codHTML += '</TD><TD WIDTH="50%" BGCOLOR="' +
banner_colorfondo_dcha + '" ALIGN="' + banner_alineacion_hor_dcha + '"
VALIGN="' + banner_alineacion_ver_dcha + '">';
  codHTML += '<FONT FACE="Verdana"
COLOR="White"><B>¿Juegos?</B></FONT>';
  codHTML += '</TD></TR></TABLE>';
 } else {
  codHTML += '<TABLE WIDTH="100%" HEIGHT="' + (banner_alto-1) + '" BOR-
DER="0" CELLSPACING="0"><TR><TD WIDTH="50%" BGCOLOR="' +
banner_colorfondo_dcha + '" ALIGN="' + banner_alineacion_hor_izda + '"
VALIGN="' + banner_alineacion_ver_izda + '">';
  codHTML += '<IMG SRC="dados.gif" WIDTH=111 HEIGHT=48 BORDER=0
ALT="">';
  codHTML += '</TD><TD WIDTH="50%" BGCOLOR="' +
banner_colorfondo_izda + '" ALIGN="' + banner_alineacion_hor_dcha + '"
VALIGN="' + banner_alineacion_ver_dcha + '">';
  codHTML += '<FONT FACE="Verdana"
COLOR="Black"><B>¿Juegos?</B></FONT>';
  codHTML += '</TD></TR></TABLE>';
 }
 codHTML += '</TD></TR></TABLE>';
 return codHTML;
 }
```

```
banner_borde_tamanyo = '1';
banner_espacio_borde_contenido = '2';

banner_colorfondo_izda = '#FFFFFF';
banner_colorfondo_dcha = '#000000';
banner_alineacion_hor_izda = 'CENTER';
banner_alineacion_ver_izda = 'MIDDLE';
banner_alineacion_hor_dcha = 'CENTER';
banner_alineacion_ver_dcha = 'MIDDLE';

...
...
//-->
</SCRIPT>
```

En el código anterior se muestra la forma de parametrizar las características relacionadas con el aspecto del *banner*, asignando los correspondientes valores iniciales. Por otra parte, en el listado 1 se muestra la función que genera el estilo dinámicamente y en ella podemos apreciar la forma de utilizar algunos de estos parámetros. Esta metodología de trabajo es muy recomendable ya que cualquier cambio de diseño que queramos realizar sobre el *banner*, una vez que hayamos finalizado todas las tareas de programación, consistirá simplemente en cambiar los valores de algunas variables.

¿Por qué generar el estilo dinámicamente en vez de poner una etiqueta *<STYLE>*? Conviene que volvamos sobre esto ya que la generación de estilos de forma dinámica es algo muy útil y que usaremos en múltiples ocasiones a la hora de realizar proyectos con *HTML* Dinámico. Existe una razón fundamental que avala la creación dinámica de estilos: permite establecer los parámetros de una capa en base a una serie de condiciones que se den a priori. Por ejemplo, el ancho de una capa (la propiedad *width*) es algo que no podemos cambiar dinámicamente con *NS4* y hay muchos casos en los que nos interesa fijar ese ancho dependiendo del ancho de la ventana del navegador.

Las variables se utilizarán para confirmar el aspecto del banner

Otra cosa que no hemos mencionado hasta el momento se refiere a un pequeño truco muy sencillo y que sin embargo resulta muy útil y elegante: la propiedad *visibility*. Inicialmente las capas siempre deben tener el valor igual a *hidden*. Sólo haremos visibles las capas cuando todos sus parámetros hayan sido configurados de manera adecuada. De esta forma evitamos la situación poco vistosa en la que el usuario está esperando a que termine de cargarse la página y aprecia en su pantalla como las cosas cambian de color, forma o contenido hasta adaptarse a su aspecto final.

EL CONTENIDO DEL BANNER: EL TRUCO DE LA TABLA

El Listado 2 contiene la función que genera el código *HTML* correspondiente al *banner*, o sea, a la capa. Esta función podrá variar bastante en función del diseño del *banner* pero en general seguirá más o menos la misma línea que la que aquí presentamos.

La función *banner_codHTML (estado)* se utilizará no sólo para generar el código *HTML* que tenga inicialmente el *banner*, sino que además generará el código en cualquier otro caso, como por ejemplo el que proponemos. Tenemos un *banner* con dos estados (que más adelante definiremos) y por lo tanto dependiendo del estado se visualizará un contenido u otro. De ahí que esta función reciba un parámetro llamado *estado*.

El ancho de una capa es algo que no podemos cambiar dinámicamente

Por otro lado observamos que hemos embebido el contenido del *banner* dentro de una tabla *HTML*. Esto no lo hemos hecho únicamente porque la tabla de la posibilidad de organizar el contenido (aunque esta ventaja también la vamos a aprovechar, por supuesto).

A pesar de que existen propiedades de las hojas de estilo que controlan el aspecto del borde de una capa, así como los márgenes internos y externos, son propiedades muy problemáticas (especialmente con *NS4*) que no se interpretan de la misma manera en *NS4* e *IE4*.

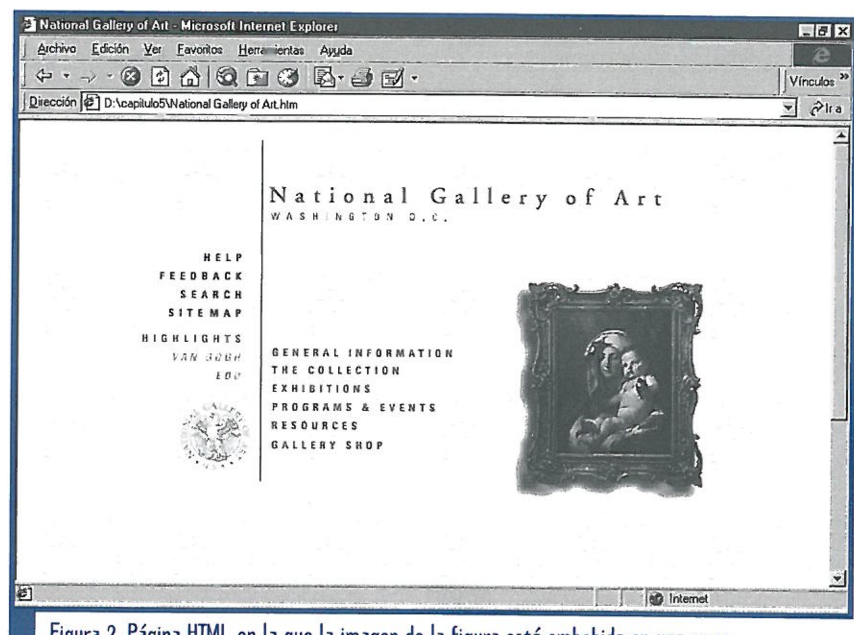


Figura 2. Página HTML en la que la imagen de la figura está embebida en una capa.

Por este motivo el borde de nuestro *banner* lo dará la propia tabla *HTML* mediante su atributo *BORDER*. Además, jugando con los atributos *CELLSPACING* y *CELLPADDING* podremos obtener distintos tipos. El resultado es algo más limitado que el propuesto por las propiedades de las hojas de estilo pero al menos se comporta del mismo modo en ambos navegadores, *NS4* e *IE4*.

Pero la razón más importante que nos ha llevado a utilizar una tabla *HTML* como elemento contenedor del *banner* es el problema que presenta *NS4* con las capas y su actualización dinámica, algo que ya vimos en los artículos pasados.

Cuando actualizamos el contenido de una capa (algo que previsible-

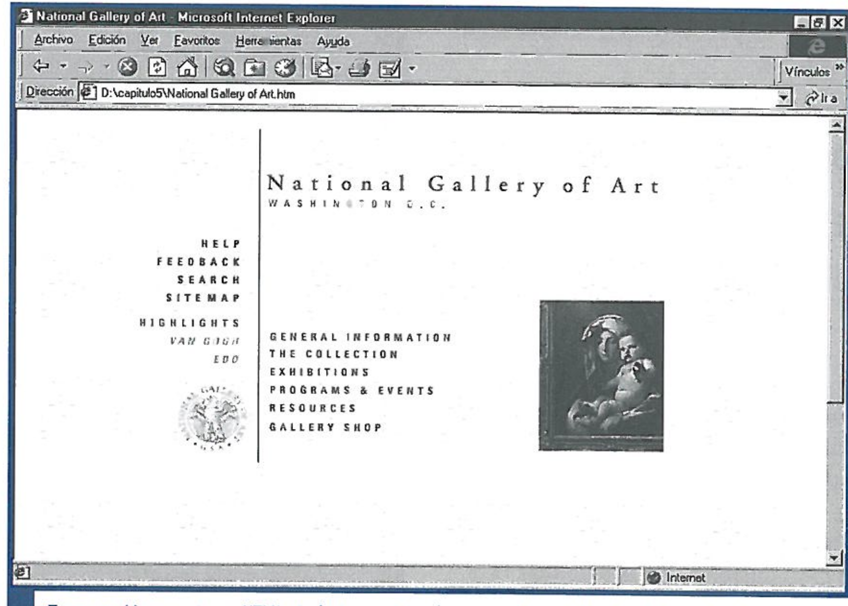


Figura 3. Misma página HTML de la figura 2 en la que se ha modificado el área de clipping recortando 20 pixels por cada lado.

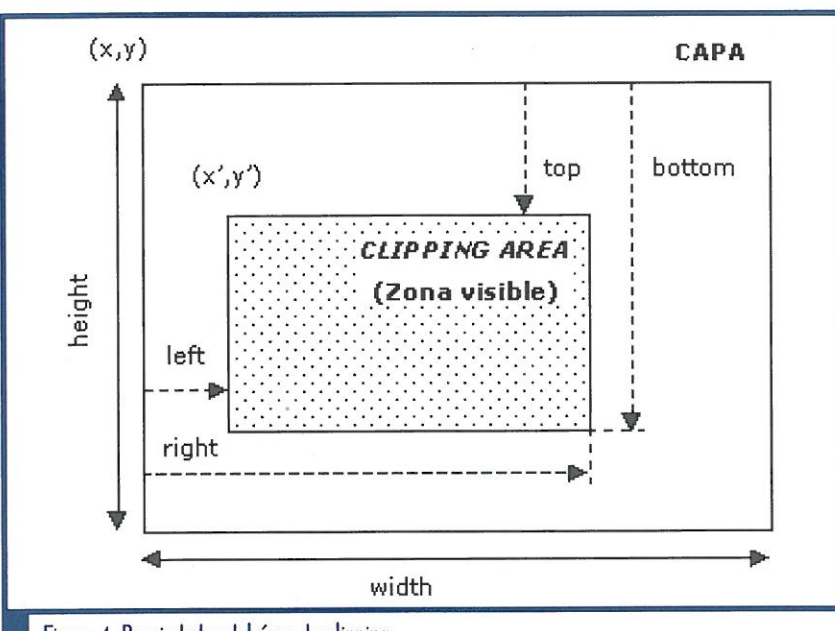


Figura 4. Propiedades del área de clipping.

mente haremos con el *banner*) con *NS4*, se pierden algunas de las propiedades de estilo de la misma, como por ejemplo la fuente y el tamaño de la letra. En algún caso hemos soslayado este problema embebiendo el código *HTML* que actualizar dentro de una etiqueta *SPAN* y "renovando", por decirlo de alguna forma, las propiedades de estilo mediante el atributo *STYLE* de la etiqueta *SPAN*. Esto mismo se puede conseguir mediante una tabla.

Los parámetros de estilo de la tabla siempre se conservarán

Para empezar podemos fijar el ancho y el alto de la tabla de forma que sea exactamente igual a la capa. Además, los parámetros de estilo de la tabla siempre se conservarán. Finalmente, la tabla es un elemento *HTML* digamos que mucho más estable, lo que se traduce en que se comportará en cuanto a sus características de diseño, de forma similar en los dos navegadores, y no variará como consecuencia de una actualización del contenido de la capa o algo así.

LA FUNCIÓN INICIAR()

Como de costumbre, todos los elementos de nuestro ejemplo se configurarán adecuadamente en la función *iniciar()* que se ejecuta cuando se produce el evento *onload* de la página. En el Listado 3 observamos que lo primero que hacemos es calcular el ancho y el alto de la ventana que aloja nuestra página *Web*. En este punto debemos recordar las diferencias entre las propiedades *window.innerWidth* y *window.innerHeight* (*NS4*), y las propiedades *document.body.clientWidth* y *document.body. clientHeight*(*IE4*).

Mientras que las primeras dependen de la ventana, las segundas dependen del cuerpo de la página. La ventana existe antes de que se cargue la página por lo que no es necesario esperar al

Listado 3. Función que inicializa todos los parámetros relacionados con el banner.

```
function iniciar() {
  if (NS4) {
    ancho_ventana = window.innerWidth;
    alto_ventana = window.innerHeight;
  } else {
    ancho_ventana = document.body.clientWidth;
    alto_ventana = document.body.clientHeight;
  }
  c_banner = new objetoCapa(“banner”);
  if (!NS4) {
    if (banner_alineacion == ‘CENTER’) {
      c_banner.left = ((ancho_ventana - banner_ancho)/2);
    } else if (banner_alineacion == ‘LEFT’) {
      c_banner.left = 0;
    } else if (banner_alineacion == ‘RIGHT’) {
      c_banner.left = (ancho_ventana - banner_ancho);
    }
  }
  c_banner.recortar(0, c_banner.getWidth() , c_banner.getHeight(),0);
  c_banner.animacion_banner = animacion_banner;
  c_banner.estado = 1;
  c_banner.actualizacion = true;
  c_banner.mostrar();
  c_banner.animacion_banner();
}
```

evento *onload* de la página para conocer sus valores, acción que tendremos que hacer en el caso de *IE4*.

Una vez que hemos calculado los valores relativos al ancho y el alto de la ventana disponible, creamos el objeto capa correspondiente al *banner* y lo posicionamos den su lugar correcto de la página. Aquí vemos de nuevo el uso de la parametrización de la que hemos hablado antes: dependiendo del valor que tenga el parámetro *banner_alineacion* y teniendo en cuenta el tamaño de la ventana, la propiedad *left* de la capa tomará un valor u otro.

Listado 4. Al objeto banner le dotamos de un método capaz de llevar a cabo la animación del mismo.

```
function animacion_banner() {
 diferencia = Math.abs(this.recorte["right"] - this.recorte["left"]);
 if (this.estado == 1) {
  if ((diferencia != 1)&&(diferencia != 0)) {
   nuevo_right = this.recorte["right"] -1;
   nuevo_left = this.recorte["left"] + 1;
  } else {
   this.estado = 0;
   this.actualizacion = !this.actualizacion;
   this.actualizarContenido(banner_codHTML(this.actualizacion));
  }
 } else {
  if (diferencia < this.getWidth()) {
   nuevo_right = this.recorte["right"] + 1;
   nuevo_left = this.recorte["left"] - 1;
  } else {
   this.estado = 1;
  }
 }
 c_banner.recortar(this.recorte["top"], nuevo_right , this.recorte["bottom"], nuevo_left);
 setTimeout('coleccionObjetosAnimados["'+ this.nombre + '"].animacion_banner()',
1);
}
```

EL ÁREA DE CLIPPING: ANIMAR LOS BANNERS

En este punto conviene recordar algo que ya estudiamos: el *área de clipping*. Ésta podía definirse como la zona visible de la capa, de tal fomra que todo lo que quede fuera de ella es como si no existiese o fuese transparente. Es decir, si hubiera algo detrás de la capa se vería aquello que estuviera detrás. Las Figura 2 muestra una página *HTML* normal en la que la imagen que se ve está contenida dentro de una capa. La Figura 3 muestra el resultado de aplicar un área de *clipping* que recorta el ancho y el alto de la capa en **20** *pixels* por cada lado. Como se puede apreciar los trozos que no se ven de la capa es como si fueran transparentes y dejan ver el fondo de atrás.

La modificación dinámica del *área de clipping* era un recurso que habíamos utilizado para simular el *scrolling* de un texto. En el artículo que nos ocupa esta vez vamos a ver otra utilidad también muy interesante: la animación de las capas. Si cambiamos de forma continua los valores del área de *clipping* de una capa, ésta varia su aspecto de forma que podemos lograr efectos múltiples. En este caso vamos a encoger y ensanchar la capa de forma que determinaremos dos momentos en el proceso de animación: cuando el *banner* se muestra en su totalidad (momento que marca-

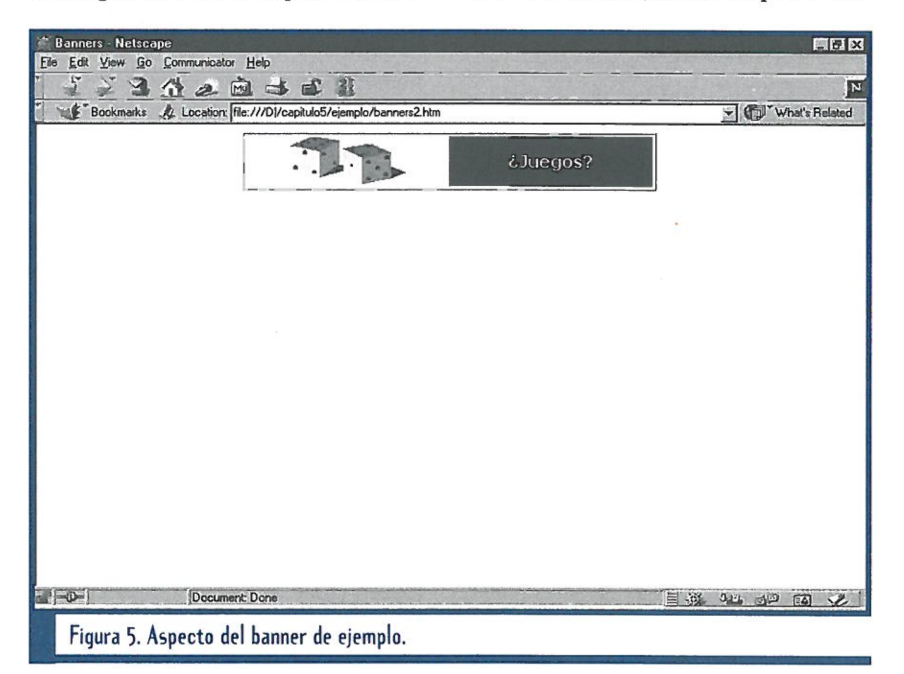


Figura 5. Aspecto del banner de ejemplo.

rá el inicio del "proceso de acortamiento") y cuando no se ve nada del *banner* (momento que marcará el inicio del "proceso de ensanchamiento"). En esos dos momentos además vamos a proceder a cambiar el contenido del *banner*, con lo que acentuaremos aún más su dinamismo.

La Figura 4 muestra los parámetros que determinan el área de *clipping* de un elemento. En la función *iniciar()* observamos que nada más crear la capa y posicionarla lo primero que hacemos es aplicarle un área de *clipping*, concretamente:

```
c_banner.recortar(0, c_banner.getWidth() ,
      c_banner.getHeight(),0);
```

En realidad si tenemos presente la Figura 4 y analizamos la línea de código anterior nos damos cuenta enseguida de que estamos aplicando un área de *clipping* que se ajusta como un guante al tamaño real de la capa, es decir, se deja la capa tal y como está. Esto se suele utilizar para inicializar los valores del área de *clipping* de forma correcta.

Las propiedades marcarán los momentos en los que debemos cambiar el contenido del banner

Una vez que hemos inicializado los valores del área de *clipping* asignamos un método y dos propiedades al objeto capa. El método será *animación_banner()* y las propiedades *estado* y *actualización*. Evidentemente las propiedades marcarán los momentos en los que debemos cambiar el contenido del *banner* y los que marcan el cambio del sentido de la animación (acortar el *banner* hasta que deje de verse y entonces empezar a ensancharlo hasta que se muestre completamente y vuelta a empezar). El método *animación_banner()* será que el que realmente lleve a cabo la animación.

EL MÉTODO ANIMACIÓN_BANNER()

El Listado 4 contiene el código correspondiente al método *animación_banner()* que es el que realiza la animación del *banner*. A grandes rasgos podemos decir que el método lo único que hacer es cambiar los valores correspondientes al área de *clipping* y se llama a sí mismo recursivamente cada cierto tiempo utilizando la función *setTimeout()*. Comparando los valores correspondientes al ancho de la capa con los valores del ancho del área de *clipping* podremos determinar cuándo ha de cambiar el sentido de nuestra animación además de detectar los momentos en los que se produce la actualización del *banner*.

La velocidad de la animación vendrá determinada por dos factores: el valor de tiempo que pongamos en la llamada a la función *setTimeout()* y el número de *pixels* que estrecharemos o aumentaremos en cada pasada. Debemos jugar con estos dos parámetroscon el fin de dar con la animación que deseemos.

Ni que decir tiene que tanto para la gestión del área de *clipping* como para la actualización dinámica del *banner* hemos hecho uso de nuestra librería *dhtml.js*. Podemos crear distintas formas de animar el *banner* simplemente creando otros métodos similares a *animación_banner()*. Por ejemplo, veamos cómo es posible hacer surgir el *banner* de la nada poco a poco como si estuviéramos estirando de sus cuatro esquinas. En este caso tendríamos que empezar con el *banner* totalmente oculto con lo que en la función iniciar debería aparecer algo así:

```
top0 = c_banner.getHeight()/2;
right0 = c_banner.getWidth()/2;
bottom0 = c_banner.getHeight()/2;
left0 = c_banner.getWidth()/2;
c_banner.recortar(top0, right0, bottom0, left0);
```

Ahora tendríamos que codificar un nuevo método *animación_banner()* de forma que modificando los valores del área de *clipping* llegara a visualizarse todo el *banner*. La cosa será tan sencilla como:

- Reduciremos el valor de *top* hasta que sea igual a **0**.
- Aumentaremos el valor de *right* hasta que sea igual al ancho del *banner*.
- Aumentaremos el valor de *bottom* hasta que tome un valor igual al largo del *banner*.
- Reduciremos el valor de *left* hasta que tome **0**.

Como se puede apreciar el objetivo consiste en llegar al mismo área de *clipping* del que partíamos precisamente en el ejemplo anterior. La animación del área de *clipping* terminará cuando ésta alcance un tamaño tal que el *banner* pueda encajar completamente en ella.

Un banner basado en HTML Dinámico puede modificarse en cualquier momento

Las posibilidades son múltiples y podemos combinar todo tipo de efectos a la hora de realizar nuestros *banners*: el *scrolling* de texto, el parpadeo de imágenes, el movimiento de las capas, la animación dinámica del área de *clipping*. El único límite es nuestra imaginación, ¡Y todo compatible!.

Demo disponible en nuestra Web.

Algunos Ratones son muy curiosos...

INFORMACIÓN

¡Proteja la información de sus aplicaciones!

DadvinaFence
Seguridad en sus Aplicaciones.

Uno de los riesgos que sufren las aplicaciones informáticas es la posibilidad de su uso por personas no autorizadas.
DadvinaFence incorpora tres componentes OCX o VCL:**ApplicationLock**, **FormLock** y **NetMessenger**, proporcionando al programador un sistema de seguridad activo fácil de integrar y totalmente configurable.

- Versiones VCL para Delphi 3.0/2.0, C++ Builder 3 y OCX para Visual Basic 5.0.
- Asignación de permisos y privilegios sobre los objetos existentes en las pantallas de las aplicaciones.
- Autentificación de usuarios mediante el uso de identificación y contraseña.
- Detección de intentos fallidos continuados de entrada al sistema,avisando al instante a los administradores de seguridad de las aplicaciones.
- Centralización de los accesos (login) a las bases de datos de la aplicación.
- Registro de entradas y salidas.
- Sistema modular fácilmente actualizable en tiempo de ejecución y de diseño.

etc...





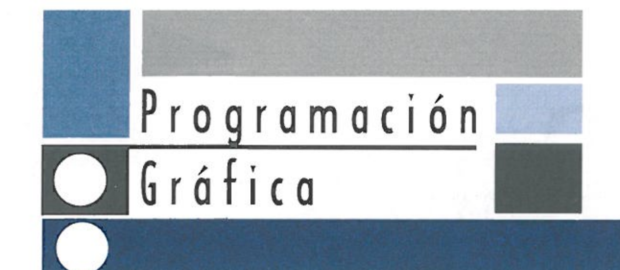
Sus clientes se lo Agradecerán

Sólo por 31.500* ptas.

San Toribio 6.
28031 Madrid.
Email: com@dadvina.com

Para más información o realizar pedidos contacte con el telefono 91 380 32 46 o en nuestra Web: www.dadvina.com.

* IVA no incluido.



Glide (IV)

Constantino Sánchez Ballesteros (constantino@nexo.es)

En esta ocasión vamos a tratar de conocer todos los detalles relacionados con la creación de los multiples efectos atmosféricos que se pueden desarrollar con una aceleradora 3Dfx.

EFECTOS ESPECIALES

G*lide* soporta diferentes tipos de efectos especiales, incluyendo *fog* (niebla), *chroma-keying* y *alpha masking*. *Fog* simula condiciones atmosféricas como niebla, llovizna o humo que oscurecen parcialmente los objetos distantes. *Chroma-keying* puede ser utilizado para crear un efecto de pantalla azul, eliminando todos los *pixels* que poseen un color específico. *Alpha masking* utiliza el *bit* de primer orden del valor *alpha* para invalidar determinados *pixels*. En este número aprenderemos a:

Mediante la inclusión de efectos atmosféricos podemos aumentar la velocidad de render

- Producir niebla utilizando *Alpha Iterator*.
- Crear tablas de niebla y utilizarlas para crear efectos atmosféricos.
- Configurar las unidades de niebla y *alpha blending* para efectuar *multi-pass fogging*.
- Usar *chroma-keying* para simular una pantalla azul (como se utiliza en el cine para superponer imágenes).
- Utilizar *alpha testing* para simular una pantalla azul.

FOG (NIEBLA)

F*og* es una técnica de renderizado que agrega realismo a las escenas generadas por ordenador haciendo que los objetos distantes desaparezcan de manera progresiva. Se trata de un término general que representa todos los efectos atmosféricos: bruma, humo de llovizna, humo, etc. Esta representación es esencial en simulaciones visuales como simuladores de vuelo que producen el efecto de visibilidad limitada. Cuando se activa el *fogging*, los objetos distantes se mezclan en el color de la niebla. Tanto el color de la niebla como su densidad (nivel al que los objetos se mezclan como una función de su distancia en referencia al que visualiza la escena) son programables.

Glide y el *hardware* de la *Voodoo* soportan operaciones de mezclado para la niebla por cada *pixel* dibujado. La unidad *fog* está separada de la unidad *alpha blending* (mezclado *alpha*), por lo que la niebla y la transparencia pueden ser aplicadas de forma simultánea. *Fog* se aplica después de las texturas y las luces y puede decaer el rendimiento de visualizado en largos escenarios *3D*: algunos objetos pueden perderse...

Fog se aplica después de la combinación de color y antes del alpha blending

La operación niebla mezcla el color *fog* (c_{fog}) con cada color de *pixel* renderizado (c_{in}) utilizando un factor de mezcla *f*, el cual se obtiene a partir de los *bits* de orden alto del valor alpha iterado o de una tabla de niebla indexada creada por el programador con el componente de *pixel* *1*/w.

La operación niebla mezcla un valor global (c_{fog}) con cada color de *pixel* renderizado (c_{in}) utilizando un factor de mezcla *f*. Un valor de *f* igual a **0** indica una densidad de niebla mínima y un valor de *f* igual a **255** indica la máxima densidad. La ecuación general de la niebla se muestra a continuación.

$$c_{out} = f\,c_{fog} + (1-f)c_{in}$$

```
void grFogMode(GrFogMode_t mode)
```

El argumento *mode* puede ser uno de los siguientes valores:

1. *GR_FOG_DISABLE*
2. *GR_FOG_WITH_ITERATED_ALPHA*
3. *GR_FOG_WITH_TABLE*
4. *GR_FOG_ADD2*
5. *GR_FOG_MULT2*

Los últimos dos modos han sido creados para facilitar aplicaciones *multi-pass fogging* (más adelante veremos qué significa este término) y se utilizan en conjunción con *GR_FOG_ WITH_ITERATED_ALPHA* ó *GR_FOG_WITH_TABLE*. La forma general de la ecuación de la niebla es la siguiente:

$$c_{out} = f\,c_{fog} + (1-f)c_{in}$$

El argumento *mode* de *grFogMode()* provee la ecuación general para una situación específica, como se muestra más adelante. Los primeros tres modos se excluyen mutuamente. Los modos *GR_FOG_ADD2* y *GR_FOG_MULT2* se utilizan de manera conjunta con *GR_FOG_WITH_ITERATED_ALPHA* ó *GR_FOG_WITH_ TABLE*.



Figura 1. En la imagen podéis comprobar el efecto que produce la niebla en un entorno 3D.

El fogging representa el efecto especial de la niebla en sus múltiples variantes

El factor de *fogging f* viene determinado por el parámetro *mode*. Si toma el valor *GR_FOG_WITH_ITERATED_ALPHA*, *f* será igual a los *bits* enteros del *alpha* iterado. Si *mode* es *GR_FOG_WITH_TABLE*, *f* será computada mediante la interpolación entre entradas de la tabla de *fog*, donde esta tabla es indexada con una representación en coma flotante del componente de pixel w. La niebla es aplicada después de la combinación de color y antes del *alpha blending* (mezcla *alpha*).

El color global de la niebla (c_{fog}) se asigna mediante la función *grFogColorValue*( ). El argumento *value* es un color *RGBA* y se especifica en el formato definido en el parámetro *cFormat* de la función *grSstWinOpen()*.

```
void grFogColorValue(GrColor_t value)
```

Tabla 1. Ecuaciones para visualizar niebla.

Si se establece mode	La ecuación de niebla será	Donde c_{in} es el color de la unidad fog, c_{out} es el resultado del *fogging*, y c_{fog} es el color de la niebla.
GR_FOG_DISABLE	$c_{out} = c_{in}$	
GR_FOG_WITH_ITERATED_ALPHA	$c_{out} = i\,c_{fog} + (1-i)c_{in}$ [	i es el *byte* de orden alto del valor *alpha* iterado.
GR_FOG_WITH_TABLE	$c_{out} = f_{fog[w]} \bullet c_{fog} + (1- f_{fog[w]}) \bullet c_{in}$	*ffog[w]* es computado mediante interpolación entre las entradas de una tabla indexada con *w-f* puede ser el *byte* de orden alto o el *alpha* iterado o computado de la tabla *fog* creada por el programador.
GR_FOG_ADD2	$c_{out} = (1-f)c_{in}$	
GR_FOG_MULT2	$c_{out} = f\,c_{fog}$	*f* puede ser el *byte* de orden alto o el *alpha* iterado o computado de la tabla *fog* creada por el programador.

FOGGING CON ALPHA ITERADO

Para crear niebla con el valor *alpha* iterado, el modo de niebla debe establecerse como *GR_FOG_WITH _ITERATED_ALPHA*. En este modo los 8 *bits* de orden superior del valor producido por el iterador *alpha* son utilizados como factor de suavizado *f*. La ecuación sería la siguiente:

$$c_{out} = i\, c_{fog} + (1-i) c_{in}$$

El siguiente extracto de código muestra el *fogging* con *alpha* iterado. No se necesita inicialización detrás del color de la niebla y su modo:

```
/* Niebla de color blanco... el color se establece
        en formato ARGB */
grFogColorValue( 0xFFFFFF);
grFogMode(GR_FOG_WITH_ITERATED_
        ALPHA);

/* los vértices poseen valores alpha que crecen
        cuando el objeto se distingue menos */
dibuja_objects();
```

FOGGING CON UNA TABLA DE NIEBLA ESPECIFICADA POR EL PROGRAMADOR

La aplicación puede agregar una tabla de niebla al hardware *mediante* la función *grFogTable()*. Para activar esta tabla, el modo de niebla debe establecerse como *GR_FOG_WITH_TABLE*. Esta tabla de niebla debería tener **64** valores de densidad del tipo *GrFog_t*, los cuales son cantidades de 8 *bits* sin signo. Un valor igual a **0** indica una densidad de niebla mínima, mientras que **255** establece la mayor densidad para el efecto especial.

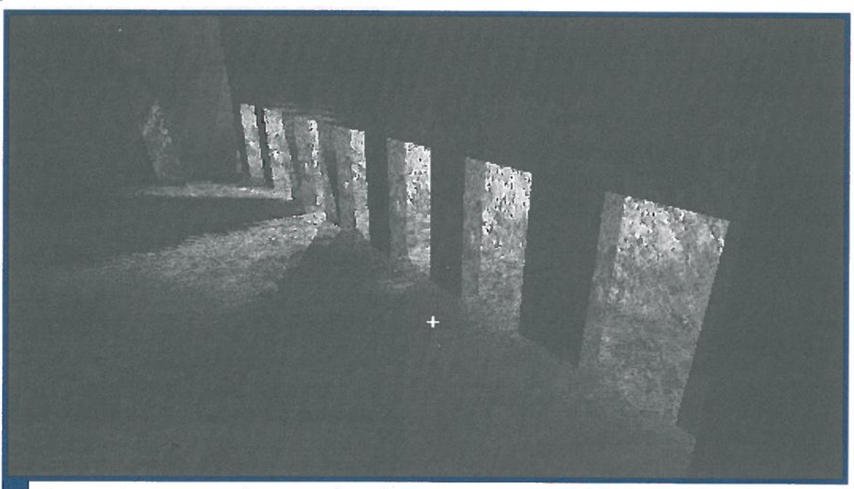


Figura 2. Los efectos de luces con alpha blending producen escenas tan espectaculares como ésta.

Esta densidad determina la cantidad de mezcla que ocurre entre el *pixel* y el color global de la niebla establecida con *grFogColorValue()*. El orden de las entradas en relación con la tabla corresponde a la distancia del observador. Las entradas de la tabla se calculan como una función de la escena *w* donde $w \ @\ 2^{i/4}$, e *i* es el índice de la tabla ($0 \leq i < 64$).

```
void grFogTable(const GrFog_t
        table[GR_FOG_TABLE_SIZE])
```

Esta función descarga una nueva tabla de valores de 8 *bits* que son vistos como valores opacos de niebla dependiendo de la profundidad. Las entradas de la tabla controlan la mezcla entre el color de la niebla y el color del *pixel*. Un valor igual a **0x00** indica que no habrá mezcla en la niebla y un valor de **0xFF** indica niebla completa.

La operación niebla mezcla el color (c_{fog}) con cada color de *pixel* renderizado (c_{in}) utilizando un factor de mezcla *f*. Cuando *grFogMode()* se establece con el valor *GR_FOG_ WITH_TABLE*, el factor *f* se computará mediante la interpolación entre las entradas de la tabla de niebla, donde esta tabla se indexa mediante una representación en coma flotante del componente *pixel w*.

$$c_{out} = f_{fog[w]} \bullet c_{fog} + (1 - f_{fog[w]}) \bullet c_{in}$$

El orden de las entradas de la tabla corresponde a la distancia obtenida en relación con el observador. El valor *w* que corresponde a la entrada de tabla *i* puede encontrarse mediante la función *guFogTableIndexToW()* con el argumento *i*.

```
guFogTableIndexToW(int i)
```

Esta función devuelve el valor *w* del ojo del observador asociado con la entrada *i* de la tabla de niebla. Puesto que las entradas de la tabla no son lineales en la variable *w*, no es correcto inicializar la tabla de niebla. *guFogTableIndexToW()* nos ayuda convirtiendo los índices de la tabla desde el punto de vista del observador *w*, y devuelve lo siguiente:

```
pow(2.0, 3.0+(double)(i>>2)) / (8-(i&3))
```

Una tabla de niebla exponencial puede generarse calculando $(1-e^{-kw}) \bullet 255$ donde *k* es la densidad de niebla y *w* es la distancia en la escena *3D*. De todos modos, es mejor normalizar la tabla para que la última entrada sea **255**.

Los siguientes trozos de código crean una tabla de niebla. El primer ejemplo muestra una tabla lineal que tiene una rampa al principio y final, con unos valores que crecen lentamente en el medio de la tabla:

```
const GrFog_t fog[63];
int i;
fog [0] = 0;
for (i=1; i<12; i++) fog[i]= fog[i-1]+ 12;
for (i=12; i<56; i++) fog[i]= fog[i-1] + 1;
for (i=56; i<63; i++) fog[i]= fog[i-1] + 7;
fog[63] = 255;
```

Esta segunda tabla representa valores exponenciales. Se calcula *w* a partir de *i* utilizando *guFogTableIndexToW()* y calculando las entradas de la tabla como *fog[i]=(1–e^-kw^)•255* donde *k* es una constante definida por el usuario, *FOG_DENSITY.*

```
#define FOG_DENSITY .5
const GrFog_t fog[GR_FOG_TABLE_SIZE];
int i;
for (i=0; i<GR_FOG_TABLE_SIZE; i++) {
        fog[i] = (1 - exp((- FOG_DENSITY) *
                guFogTableIndexToW(i))) * 255;
}
fog[GR_FOG_TABLE_SIZE] = 255;
```

FOGGING CON 1/W Y UNA TABLA DE NIEBLA

En el siguiente trozo de código se asume que ha sido definida una tabla de niebla. La cargamos mediante el método *grFogTable()*. De esta forma un color de niebla queda definido así como su modo de niebla. Lo único que hace falta es dibujar la escena.

```
const GrFog_t fog[GR_TABLE_SIZE];
int i;

/* Cargamos la tabla de niebla */
grFogTable(fog);

/* Establecemos un color para la niebla – qué tal
        negro para el humo? */
grFogColorValue(0);

/* Establecemos el modo para la tabla de niebla
        */
grFogMode(GR_FOG_ WITH_TABLE);

/* dibujamos la escena */
```

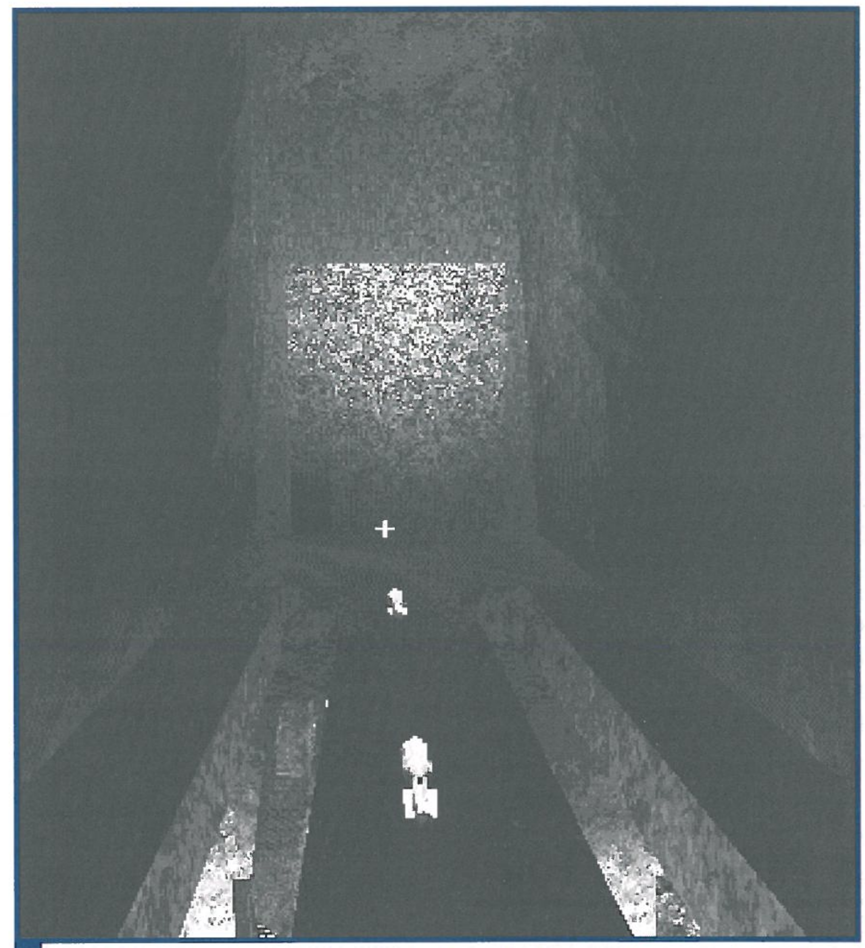


Figura 3. Los resultados que producen las las luces volumétricas realmente son efectos de niebla pero que radian luz.

GENERANDO UNA TABLA DE NIEBLA AUTOMÁTICAMENTE

La librería de utilidades de *Glide* incluye tres importantes rutinas que generan tablas de niebla cque contienen diferentes e interesantes características:

```
void guFogGenerateExp(GrFog_t fogTable[GR_
        FOG_ TABLE_ SIZE], float density )
```

Esta función genera una tabla exponencial acorde a la ecuación:

$$e^{-density \bullet w}$$

donde *w* es la coordenada del punto de vista del observador *w* asociada con la entrada de la tabla. La tabla resultante es copiada dentro de *fogTable*. La tabla es normalizada (escalada) hasta que la última entrada tenga un valor máximo de **255**.

```
void guFogGenerateExp2(GrFog_t
        fogTable[GR_FOG_TABLE_ SIZE],
                float density )
```

Genera una tabla exponencial acorde a la siguiente ecuación:

$$e^{-(density \bullet w)\ (density \bullet w)}$$

donde c_i es el color de la unidad de niebla en el paso *i*.

Una vez concluida la primera pasada nos introducimos en la segunda. Ésta se puede generalizar como múltiples pasos por inducción. Nosotros deseamos obtener la siguiente ecuación:

$$c_{dst} = Fog(c_1 + c_2) = fc_{fog} + (1-f)(c_1 + c_2)$$

Para la primera pasada eligiremos tanto *GR_FOG_WITH_TABLE* o *GR_FOG_WITH_ITERATED_ALPHA* para el modo de niebla como los factores de origen y destino a *GR_BLEND_ONE* y *GR_BLEND_ ZERO*, respectivamente.

Alpha blending reserva un número variable de colores para crear su transparencia

Después de la primera pasada obtendremos:

$$\begin{aligned} c_{dst} &= 1 \bullet Fog(c1) + 0 \bullet cdst \\ &= Fog(c_1) \\ &= fc_{fog} + (1-f)c_1 \end{aligned}$$

Para la segunda pasada agregamos *GR_FOG_ADD2* al modo de niebla, provocando que el mezclado final sea suprimido (si olvidamos hacer esto, c_{fog} se creará por duplicado). Es necesario establecer los factores origen y destino de *alpha blending* como *GR_BLEND_ ONE* y *GR_BLEND _ONE*, respectivamente:

$$Fog(c_2) = (1-f)c_2$$

$$\begin{aligned} c_{dst} &= 1 \bullet cin + 1 \bullet c_{dst} \\ &= (1-f)c_2 + (fc_{fog} + (1-f)c_1) \\ &= fc_{fog} + (1-f)(c_1 + c_2) \end{aligned}$$

A continuación vamos a poder ver en la práctica el proceso para la creación de niebla *multipass* aditiva (se asume que ya se ha definido una tabla de niebla):

```
const GrFog_t fog[GR_TABLE_SIZE];
int i;

/* Cargamos la tabla de niebla */
grFogTable(fog);

/* Asignamos un color de niebla */
grFogColorValue(0);

/* Establecemos modo a la tabla de niebla */
grFogMode(GR_FOG_WITH_TABLE);
grAlphaBlendFunction(GR_BLEND_ONE,
      GR_BLEND_ZERO,
            GR_BLEND_ONE,
      GR_BLEND_ZERO);

/* Dibujamos la primera pasada */
...

/* Asignamos modo a la tabla de niebla */
grFogMode(GR_FOG_WITH_TABLE |
      GR_FOG_ADD2);
grAlphaBlendFunction(GR_BLEND_ONE,
      GR_BLEND_ONE,
      GR_BLEND_ONE, GR_BLEND_ZERO);

/* Dibujamos la segunda pasada */
      ...
```

CHROMA-KEYING

Cuando se activa el *chroma-keying*, los valores de color se comparan con un valor de referencia global establecido con la función *grChromakey-Value()*. Si el color del *pixel* es el mismo que el valor de referencia de *chroma-key*, el *pixel* no se dibujará.

La comparación de *chroma-key* tiene lugar antes de la función de combinación de color; el otro color seleccionado por la función de combinación de color será el único comparado. Por defecto, *chroma-keying* está desactivado.

Chroma-keying resulta muy útil para ciertos tipos de animaciones de *sprites* o pantallas azules en texturas. Sólo se reserva un color para la transparencia del *chroma-key*, mientras que *alpha blending* reserva un número variable de colores para crear su transparencia.

```
void grChromakeyMode( GrChromakeyMode
      _t mode )
```

Para activar o desctivar el *chroma-key* se utiliza la función *grChromakey-Mode()*.

La técnica de chroma keying sirve para definir un color determinado como transparente

El argumento *mode* especifica cuándo se debe activar o desactivar el *chroma-key*. Los valores válidos son *GR_CHROMAKEY_ENABLE* y *GR_CHROMAKEY_DISABLE*.

```
void grChromakeyValue(GrColor_t value)
```

La función *grChromakeyValue()* asigna el valor global de referencia de *chroma-key* como un valor *RGBA* empaquetado como el formato especificado en el parámetro *cFormat* de *grSstWinOpen()*.

SIMULANDO UNA PANTALLA AZUL CON CHROMA-KEYING

Una pantalla azul es un mecanismo de composición utilizado sobre todo en el mundo del vídeo y cine donde una segunda escena superpone todos los pixels azules de la primera. A continuación veremos un ejemplo de cómo se puede programar el *chroma-keying* con *Glide*:

Tabla 3. Funciones de Alpha testing.

Alpha testing es una técnica donde el valor alpha entrante es comparado con un valor de referencia, ignorando el pixel si falla el test. Este test puede ser elegido por el usuario, pudiendo escoger las siguientes opciones:

Si func es...	Función de comparación
GR_CMP_NEVER	Nunca pasa el test.
GR_CMP_LESS	Lo pasa si el valor α producido por la unidad de combinación alpha es menor que el valor constante alpha utilizado como referencia.
GR_CMP_EQUAL	Lo pasa si el valor α producido por la unidad de combinación alpha es igual que el valor constante alpha utilizado como referencia.
GR_CMP_LEQUAL	Lo pasa si el valor α es menor o igual que el valor α de referencia.
GR_CMP_GREATER	Lo pasa si el valor α es mayor que el valor α de referencia.
GR_CMP_NOTEQUAL	Lo pasa si el valor α no es igual que el valor α de referencia.
GR_CMP_GEQUAL	Lo pasa si el valor α es mayor o igual que el valor α de referencia.
GR_CMP_ALWAYS	Siempre pasa el test.

```
/* Dibujamos el background */
draw_weather_map();

/* Activamos el chroma-keying */
grChromakeyMode(GR_CHROMAKEY_
        ENABLE);

/*Establecemos el color de referencia –
        asumimos el formato ARGB */
grChromakeyValue(0xFF);

/* Dibujamos la escena insertada sobre el
        background */
draw_weatherman();
```

ALPHA TESTING

Se trata de un a técnica que permite aceptar o ignorar un *pixel* en función de su valor *alpha*.

Podemos indicar el grado de opacidad mediante el valor alpha

El valor *alpha* entrante (valor de salida de la unidad de combinación *alpha*) se compara con un valor de referencia y se acepta o ignora en base a una función de comparación.

Alpha testing se utiliza para crear transparencias sobre los valores alpha de los pixels

Si creamos una textura con áreas transparentes y opacas, podemos indicar ese grado de opacidad mediante el valor *alpha*.

Debemos establecerlo a **0** si el contenido de la textura es transparente, y asignar el valor **1** en caso contrario (contenido opaco). Con un valor alpha de **.5** (o cualquier número mayor que **0**) y una función de comparación, el contenido transparente de la textura será ignorada y los *pixels* de destino serán visualizados.

Los *pixels* entrantes pueden ser ignorados basándose en una comparación entre su valor *alpha* y un valor de referencia global.

La naturaleza de esta comparación es definida por el usuario a través de la función *grAlphaTestFunction()*. Esto es útil para crear algunos efectos como mapas de texturas parcialmente transparentes.

Para desactivar *alpha testing* estableceremos la función de *alpha test* como *GR_CMP_ALWAYS*. El valor global de *alpha test* se asigna con la llamada a la función *grAlphaTestReferenceValue()*.

Dado que *alpha testing* no requiere guardar valores *alpha* (*alpha buffer*) siempre estará disponible salvo que se use *"depth ó triple buffering"*.

```
void grAlphaTestFunction(GrCmpFnc_t func)
```

El valor alpha entrante es comparado con el valor constante *alpha* utilizando la función especificada por el parámetro *func*. Ahora vemos que los posibles valores para el parámetro son los siguientes:

Utilizando Alpha testing conseguimos simular una pantalla azul

Alpha testing es creado sobre todos los *pixels* dibujados, incluyendo aquellos que resultan de la conversión de puntos, líneas, y triángulos.

Tabla 2. Opciones para el atributo BORDERSTYLE.

Valor	Significado
normal	Estilo de borde normal de una ventana.
complex	Estilo de borde consistente en una combinación de los estilos raised y sunken.
raised	Estilo de borde 3D sobresaliente.
static	Estilo de borde 3D. Este estilo es que se utiliza típicamente en aquellas ventanas que no esperan datos del usuario.
sunken	Estilo de borde 3D hundido.

Este atributo determina si aparece el título de la aplicación *HTML* (*HTA*) en la barra de título de la ventana. La propiedad *caption* es sólo de lectura y toma una valor por defecto igual a **yes**.

commandLine

Esta propiedad guarda la ruta que puede utilizarse para ejecutar la aplicación *HTML* (*HTA*) desde la línea de comandos. Es sólo de lectura y no toma ningún valor por defecto.

ICON | icon

Este atributo contiene la localización del icono asociado a la aplicación *HTML* (*HTA*). La propiedad *icon* es sólo de lectura y toma un valor por defecto igual al icono de aplicación del sistema. Reconoce iconos estándar de *Windows* de 32x32 *pixels* (*.ICO*).

MAXIMIZEBUTTON | maximizeButton

Este atributo determinará si aparece o nó el botón de maximización en la barra de título de la aplicación *HTML* (*HTA*). La propiedad *maximizeButton* es sólo de lectura y toma un valor por defecto igual a **yes**.

MINIMIZEBUTTON | minimizeButton

Este atributo determinará si aparece o no el botón de minimización en la barra de título de la aplicación *HTML* (*HTA*). La propiedad *minimizeButton* es sólo de lectura y toma un valor por defecto igual a **yes**.

SHOWINTASKBAR | showInTaskBar

Este atributo determina si la aplicación *HTML* (*HTA*) aparecerá en la barra de tareas de *Windows* y en la lista de aplicaciones que se muestra cuando pulsamos **ALT+TAB**. La propiedad *showInTaskBar* es sólo de lectura y va a tomar un valor por defecto igual a **yes**.

SINGLEINSTANCE | singleInstance

En una aplicación HTML el código JavaScript se clasifica dependiendo de su finalidad

Este atributo determina si permitimos o no que puedan ejecutarse concurrentemente varias instancias de la aplicación *HTML* (*HTA*). La propiedad *singleInstance* es sólo de lectura y toma un valor por defecto igual a **no**.

VERSION | version

Este atributo guarda la versión de la aplicación *HTML* (*HTA*). La propiedad *versión* es sólo de lectura y por defecto toma un valor igual al de la cadena vacía.

WINDOWSTATE | windowState

Este atributo determina el tamaño inicial de la ventana de la aplicación *HTML* (*HTA*). La Tabla 3 muestra los posibles valores. La propiedad *window State* es de lectura y escritura y toma un valor por defecto igual a **normal**.

TRUSTED

Este atributo determina si los contenidos de un marco (*FRAME* ó

Figura 2. "Navegador de directorios": carpetas que cuelgan del directorio actual.

IFRAME) pertenecientes a la aplicación *HTML* (*HTA*) son seguros o no. Si toma un valor igual a **no**, se aplicarán al contenido de ese marco las mismas reglas de seguridad que se aplican a cualquier página *Web*. Este atributo va a tomar un valor por defecto igual a **yes**.

UNA APLICACIÓN HTML (HTA): NAVEGADOR DE DIRECTORIOS

Nada mejor que un ejemplo para demostrar todas las cualidades de las aplicaciones *HTML* (*HTA*). Vamos a realizar un navegador de directorios que permitirá recorrer el árbol de directorios de nuestro sistema, crear o borrar directorios y ver las propiedades de los mismos.

La figura 2 muestra el aspecto que tendrá nuestra aplicación *HTML* (*HTA*).

El atributo ICON nos permite asociar iconos estándar Windows de 32 x 32 pixels

Antes de empezar es preciso poner de manifiesto que es necesario tener unos ciertos conocimientos de *HTML, Javascript y HTML Dinámico*. Cuanto mayor sea nuestra experiencia en estas tecnologías más eficientes y atractivos podrán resultar nuestros interfaces.

En cualquier caso no resulta excesivamente complicado entender el código de este ejemplo y se puede utilizar como punto de partida para realizar todo tipo de aplicaciones

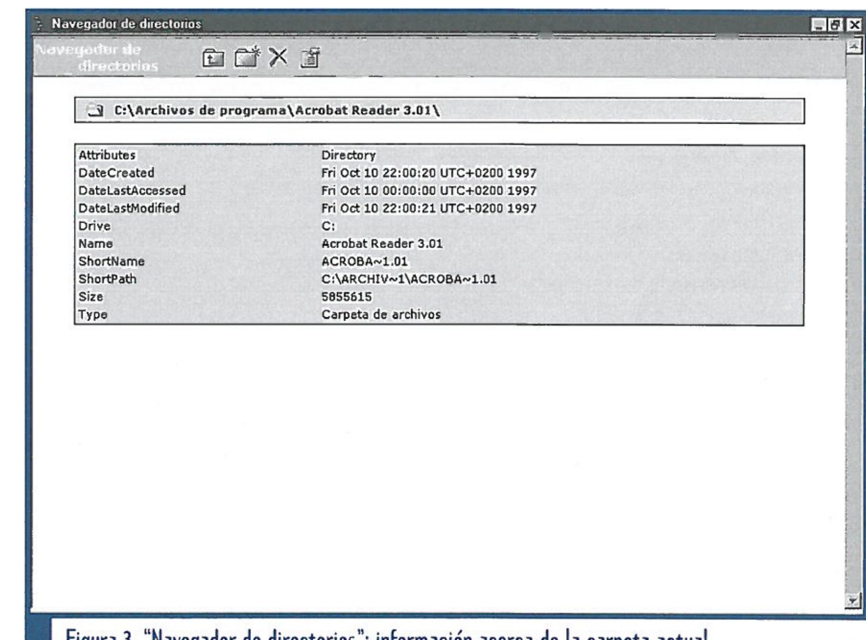


Figura 3. "Navegador de directorios": información acerca de la carpeta actual.

HTML (*HTA*) complejas basadas en los *scripts WSH* que hemos venido mostrando a lo largo de toda esta serie de artículos.

Listado 2. Funciones que realizan las operaciones más importantes de la aplicación "Navegador de directorios".

```
function lista_directorios(dir) {
  var fso, f, fc, s;
  fso = new ActiveXObject(“Scripting.FileSystemObject”);
  f = fso.GetFolder(dir);
  fc = new Enumerator(f.SubFolders);
  resultado = new Array();
  i=0;
  for (; !fc.atEnd(); fc.moveNext()) {
    resultado[i] = fc.item();
    i++
  }
  resultado.sort(ordenar);
  return(resultado);
}

function crear_directorio(dir) {
  fso = new ActiveXObject(“Scripting.FileSystemObject”);
  fso.CreateFolder(dir);
}

function borrar_directorio(dir) {
  fso = new ActiveXObject(“Scripting.FileSystemObject”);
  dir = dir.substring(0, dir.lastIndexOf(‘\\’));
  fso.DeleteFolder(dir);
}
```

Listado 3. Función que prepara la presentación del interfaz.

```
function iniciar() {
  barrabotones.style.width = document.body.clientWidth;
  barrabotones.style.top = 1;
  barrabotones.style.left = 0;
  barrabotones.style.background = "#C6C6C6";
  barrabotones.style.visibility = "visible";

  directorio.style.top = barrabotones.offsetHeight + separacion_vertical;
  directorio.style.left = directorio_margen;
  directorio.style.width = document.body.clientWidth - 2*directorio_margen;
  directorio.style.background = directorio_colorfondo;
  directorio.style.visibility = "visible";

  directorios.style.top = parseInt(directorio.style.top,10) + directorio.offsetHeight + sepa-
racion_vertical;
  directorios.style.left = directorio_margen;
  directorios.style.width = document.body.clientWidth - 2*directorio_margen;
  directorios.style.background = directorio_colorfondo;
  directorios.style.visibility = "visible";
}
```

LA INTERFAZ

Nuestra página *HTML* va a tener tres capas definidas mediante la etiqueta *DIV*: **barrabotones, directorio** y **directorios.** La primera de ellas contendrá la barra de herramientas mediante la cual el usuario va a manejar la aplicación. Está formada por una serie de imágenes que cambian dependiendo del tipo de evento: *onmouseover* hace que la imagen se resalte y tome color; *onmouseout* hace que la imagen retorne a su estado inicial y *onclick* ejecutaría la acción correspondiente.

La etiqueta HTA:APPLICATION y sus atributos indican a la ventana como debe comportarse

El botón **arriba** hará que subamos un directorio en la jerarquía con respecto al directorio actual que se está mostrando en la capa **directorio.** El botón **crear** muestra una ventana en la que podremos introducir el nombre del directorio que queremos crear dentro del directorio actual. El botón **eliminar** borrará el directorio actual y todo su contenido. Finalmente el botón **propiedades** mostrará toda la información disponible acerca del directorio actual (ver figura 3).

El esquema de funcionamiento de la aplicación es bien sencillo. Nada más empezar el programa muestra en el directorio actual el directorio donde se ubica el archivo *HTA* de la propia aplicación. A partir de esta momento podemos navegar por los directorios del disco, ya sea ascendiendo en la jerarquía mediante el botón "arriba" o haciendo clic en alguno de los directorios que se muestra en la lista de directorios y que se encuentran dentro del directorio actual.

Dentro de la aplicación *HTML* (*HTA*) podemos dividir las funciones del código *JavaScript* en dos grupos: aquellas que se encargan de realizar todas las tareas relacionadas con la interfaz (posición de los elementos, funcionamiento de los botones, etc.) y las que manejan los directorios (mostrar la información, crear y/o borrar, etc.). Aunque muchas veces esta línea divisoria es difusa vamos a analizar el código de esta forma ya que parece la más clara.

FUNCIONES QUE MANEJAN LOS DIRECTORIOS

El listado 2 muestra las tres funciones principales dentro de este grupo. Para aquellos lectores que hayan seguido la serie de artículos acerca de *Windows Scripting Host* no revestirán ninguna complicación ya que se trata simplemente de meter en funciones algunos de los *scripts* que antes habíamos realizado como programas autónomos.

El HTML resulta ideal para realizar elegantes interfaces para nuestros scripts

- *lista_directorios(dir)*

  Esta función devuelve un *array* con una lista de los directorios que se encuentran dentro del directorio que se le pasa como argumento. La única peculiaridad es que antes de devolver el *array* lo ordenamos alfabéticamente. Para ello hemos creado la función *ordenar(a,b)* que ordena los elementos no distinguiendo entre mayúsculas y minúsculas.

- *crear_directorio(dir)*

  Esta función crea un directorio dentro del directorio actual con el nombre especificado por su parámetro.

- *borrar_directorio(dir)*

  Cuando el usuario pulsa el botón **eliminar** para borrar el directorio actual

Tabla 3. Opciones para el atributo WINDOWSTATE.

Valor	Significado
normal	La ventana de la aplicación HTML (HTA) tendrá el tamaño por defecto asignado a Internet Explorer.
minimize	La aplicación HTML (HTA) se inicializará minimizada.
maximize	La aplicación HTML (HTA) se inicializará maximizada.

Tabla 4. Direcciones de interés.

HTML Applications Reference
http://www.eu.microsoft.com/workshop/author/hta/reference/htaref.asp
HTML Applications Overview
http://www.eu.microsoft.com/workshop/author/hta/overview/htaoverview.asp
Microsoft Internet Explorer 5.0 Home
http://www.microsoft.com/windows/ie/ie5/default.htm

y todo su contenido se invoca a esta función. Como la cadena de texto con el directorio que pasamos como argumento termina con el carácter separador de directorios \ se lo quitamos.

- *directorio_informacion(dir)*
  Esta función muestra, en el mismo área donde se muestran todos los directorios que están dentro del directorio actual, todos sus datos relativos.

- *contenido_directorio(dir)*
  Cada vez que es necesario actualizar el directorio actual llamaremos a esta función que se encargará de generar el código *HTML* necesario.

- *contenido_capa_directorios(dir)*
  Esta función resultaría ser la equivalente a la anterior pero en cambio lo que actualiza es la lista de directorios que se encuentran dentro del directorio actual.

LAS FUNCIONES RELACIONADAS CON LA INTERFAZ

En este grupo se engloban todas aquellas funciones que manejan la interfaz de la aplicación. El listado 3 muestra la función iniciar(), que es la encargada de preparar la presentación de la interfaz. Básicamente se encarga de posicionar correctamente los elementos en la página *HTML*.

- *hacerBotonMouseOver(img), hacerBotonMouseOut(img), hacerBotonMouseDown(img), hacerBotonMouseUp(img)*
  Estas funciones son los manejadores de los eventos que desencadenan las acciones sobre los botones de la barra de herramientas.

- *hacerEnlaceClick(enlace)*
  Cada vez que el usuario haga clic sobre alguno de los directorios que se muestran en la lista de directorios esta función se encargará de actualizar la capa correspondiente al directorio actual, así como la que contiene la lista de directorios.

Finalmente tenemos dos funciones más. La función *directorio_inicial()* se encarga de averiguar cuál es el directorio donde se ubica el archivo *HTA* correspondiente a la aplicación. Este directorio será el que tomaremos como directorio inicial. Para ello consultaremos la propiedad *window.location.href*.

WINDOWSTATE determina el tamaño inicial de la ventana de la aplicación HTML (HTA)

El inconveniente que tiene el valor que devuelve esta propiedad es que utiliza el carácter \ como separador de directorios y que además contiene el protocolo *file:///*. Algo similar surgirá a lo largo del *script* y para ello hemos creado la función *arreglar(cadena)* que se encarga de generar una referencia correcta.

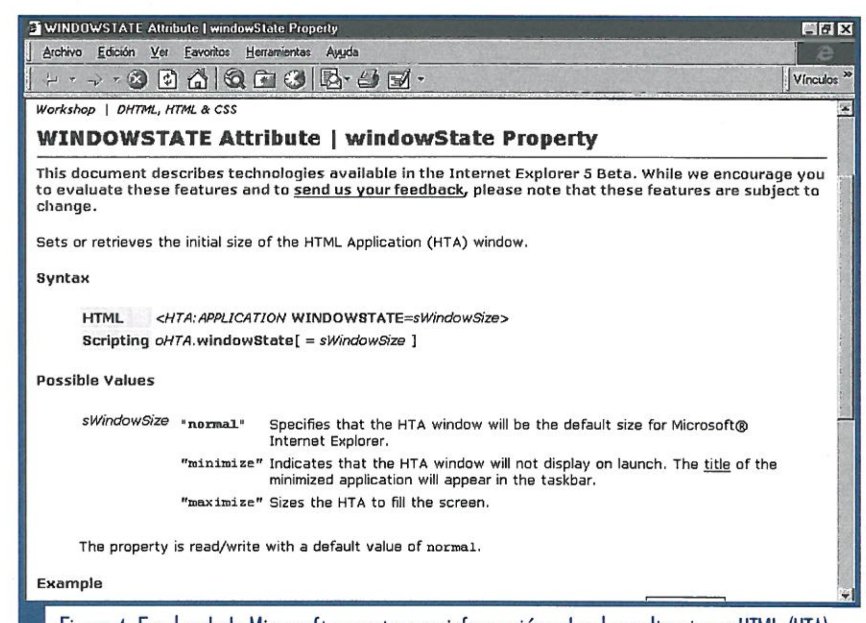
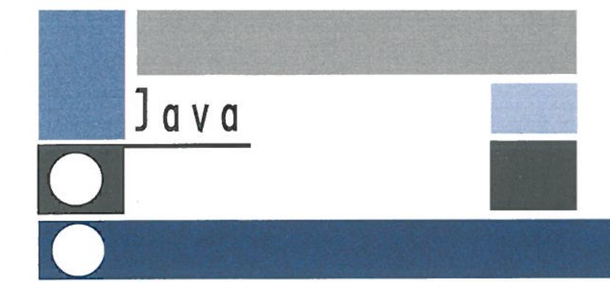


Figura 4. En el web de Microsoft encontramos información sobre las aplicaciones HTML (HTA).



Tratamiento de imágenes en Java (y II)

Javier Sanz Alamillo (jsanza@teleline.es)

A lo largo de este artículo se mostrará como realizar un tratamiento de imágenes basado en la *API Java 2D*, todo ello de una forma práctica y eficaz. Si en el anterior capítulo aprendimos los fundamentos básicos de la *API*, en éste se explicará la manera de realizar rotaciones, ampliaciones, composiciones y sencillos filtros sobre cualquier imagen.

Gracias a la ampliación del *AWT* con la *API de Java 2D* incluido en la versión *JDK 1.2*, se pueden manipular las imágenes de una forma más simple y potente, pudiéndose realizar efectos tales como *antialiasing*, *clipping*, rotaciones, filtros, etc., como si de una tarea normal se tratara.

INTRODUCCIÓN

El tratamiento de imágenes es posible gracias a que éstas se presentan como un *raster* de *pixels* con unos determinados colores. Para ello, disponemos de la clase *java.awt.image.BufferedImage* con la que se pueden realizar efectos básicos como ampliaciones, reducciones, rotaciones, etc. Además, mediante *java.awt.image.BufferedOp* podremos realizar filtros básicos para aplicar a nuestras imágenes, tales como reducción de ruido, borroso, detección de bordes, etc.

TRANSFORMACIONES

Usando la *API Java 2D* es posible realizar muchas transformaciones sobre las imágenes. Además, éstas se pueden aplicar igualmente sobre textos, figuras geométricas y demás objetos gráficos *2D*.

Las imágenes son tratadas como si tuvieran su propio *Image Space* ó Espacio de Imagen. Este espacio, el tercero posible, es un sistema de coordenadas en el cual las imágenes son transformadas al *User Space* y de aquí al *Device Space* que se mostrará por el dispositivo deseado, ya sea un monitor, una impresora, etc.

Las transformaciones se efectúan mediante un objeto *AffineTransform* que se utiliza para realizar las transformaciones del *User Space* al *Device Space*, por lo que también se utiliza para realizar la misma tarea entre el *Image Space* y el *User Space*. En el listado 1 podemos ver un ejemplo en el que se muestra la utilización de la transformación del *User Space* al *Device Space*, mediante la visualización, ampliación y rotación-reescalado de una imagen.

Se ejecuta y obtenemos el resultado que aprecia en la figura 1.

Si analizamos el código del listado 1 de forma detallada observamos que al prinicipio se aplica la opción de *antialiasing*, que básicamente permite que todos los resultados de transformaciones ofrezcan una presencia mejor, en la que no aparezcan unas esquinas con cuadros demasiados visibles, es decir, que la imagen resultante esté bien definida.

```
RenderingHints qHints = new RenderingHints (
   RenderingHints.KEY_ANTIALIASING,
   RenderingHints.VALUE_ANTIALIAS_ON );
qHints.put ( RenderingHints.KEY_RENDERING,
   RenderingHints.VALUE_RENDER_
      QUALITY );
g2.setRenderingHints ( qHints );
```

A continuación visualizamos la imagen tras desplazarla del origen de coordenadas del *Device Space*, mediante el siguiente código, por lo que ahora el *User* y el *Device Space* no coinciden:

```
AffineTransform at = new AffineTransform ();
at.translate ( 20f, 0f); // Cambio User Space
g2.transform (at);
g2.drawImage ( img, at, this );
```

En la siguiente transformación (*conRes*) se realiza una ampliación de la imagen, que se basa en este código:

```
AffineTransform conRes= new AffineTransform ();
conRes.setToTranslation ( 150f, 10.f );
g2.transform (at);
conRes.scale ( 1.4f, 1.4f );
g2.drawImage ( img, conRes, this );
```

Utilizamos una nueva transformación para no perder el *User Space* actual, de ahí la utilización de un *Image Space*. Mediante *setToTranslation* situamos la magen, realizamos la transformación para adaptarla según el *User Space* y utilizamos *scale(x,y)* para realizar la ampliación de la imagen, tanto en alto como ancho. A continuación, dibujamos la imagen según su *Image Space*. Para finalizar, realizamos una rotación y un reescalado de la imagen, mediante el código siguiente:

```
at.translate ( 0f, 120f);
g2.transform ( at );
conRes.setToShear ( 0.1f, 0.4f );
g2.drawImage ( img, conRes, this );
```

Gracias a *setToShear(x,y)* realizamos la rotación y el reescalado, reutilizando el *Image Space* de la imagen.

Listado 1. Ejemplo de transformación del Uses Space al Device Space.

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.applet.*;

public class ejemplo extends Applet {
  public void paint ( Graphics g ){
Image img = getImage ( getDocumentBase(), "duke.gif");

   Graphics2D g2 = (Graphics2D) g ;
   g2.setBackground(Color.white);
   RenderingHints qHints = new RenderingHints (
        RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON );
qHints.put ( RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY );
   g2.setRenderingHints ( qHints );

   // Visualiza la imagen tras desplazarla
   AffineTransform at = new AffineTransform ();
   at.translate ( 20f, 0f);
   g2.transform (at);
   g2.drawImage ( img, at, this );

   // Ampliacion y a la derecha
   AffineTransform conRes= new AffineTransform ();
   conRes.setToTranslation ( 150f, 10.f );
   g2.transform (at);
   conRes.scale ( 1.4f, 1.4f );
   g2.drawImage ( img, conRes, this );

   // Reescalado y abajo
   at.translate ( 0f, 120f);
   g2.transform ( at );
   conRes.setToShear ( 0.1f, 0.4f );
   g2.drawImage ( img, conRes, this );
}
}
```

COMPOSICIÓN

Cuando dos imágenes se solapan o superponen es necesario determinar qué colores son tratados en el solapamiento de los *pixels*. Por ejemplo, si tenemos un rectángulo rojo y un rectángulo azul que se solapan, hay que determinar que pixels se mantienen como rojo, cuáles como azul y cuáles son una combinación de ambos y en qué propor-

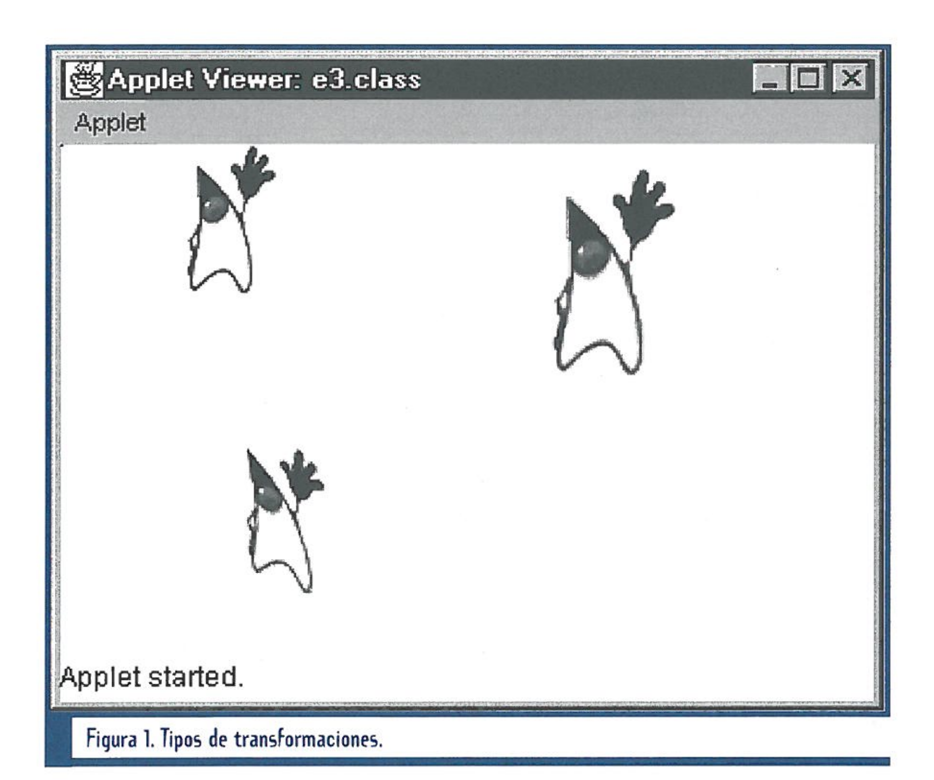


Figura 1. Tipos de transformaciones.

ción. El proceso de determinar qué color se asigna a la zona que se solapa se denomina composición. En la figura número 2 se muestra una escala de todo el posible proceso de composición.

Para realizar una composición se especifica el tipo de la misma y se crea un objeto del tipo *AlphaComposite* en un contexto gráfico *2D*, invocando el método *setComposite*. La clase *AlphaComposite* implementa la interfaz *Composite* permitiendo realizar un conjunto de diferentes composiciones.

La opción más utilizada es la del tipo *SRC_OVER* ó *Porter-Duff*, que indica que el nuevo color (el color superior) debe ser mezclado en una determinada proporción con el color destino de forma que aparenten cierta mezcla (con o sin solapamiento). Veamos un ejemplo.

El resultado de ejecutar el programa se puede apreciar en la figura 3.

Como se observa, inicialmente se dibuja una estrella mediante las siguientes sentencias:

```
GeneralPath gp = new GeneralPath(General-
      Path.WIND_NON_ZERO);
gp.moveTo( -100.0f, -25.0f );
gp.lineTo( +100.0f, -25.0f );
gp.lineTo( -50.0f, +100.0f );
gp.lineTo( + 0.0f, -100.0f );
gp.lineTo( +50.0f, +100.0f );
gp.closePath();
```

A continuación se define la composición, del tipo *SRC_OVER* y al **50%** de mezcla:

```
AlphaComposite calfa =
      AlphaComposite.getInstance(
 AlphaComposite.SRC_OVER, 0.5f);
g2.setComposite ( calfa );
```

Con *calfa* se define una mezcla al **50%** ó **0.5**, aunque también se podría definir que la imagen destino sea opaca, situando *calfa* al **0%** ó **0.0f** y también al revés, si ponemos *calfa* al **100%** ó **1.0f** se visualizará como una imagen sobre otra, pero sin apariencia de mezcla.

Para finalizar, se sitúa una imagen en el centro de la estrella y se visualiza:

```
g2.translate ( -30.0f, -30.0f);
g2.transform (at);
g2.drawImage ( img, at, this );
```

Como se observa, al ser una mezcla al **50%**, la imagen central aparentemente parece estar mezclada con la imagen de destino o fondo.

TRATAMIENTO ESPACIAL

El tratamiento de imágenes más realizado y conocido, muy utilizado en programas como *Adobe PhotoShop*, es el tratamiento espacial o *spatial filtering*, también denominado *Convolution*. Mediante este proceso, se calcula el color de un determinado *pixel* en función de su color y del color de los que le rodean, aplicando una simple operación matemática en función de la cual se obtendrá un color resultado para el *pixel* en cuestión.

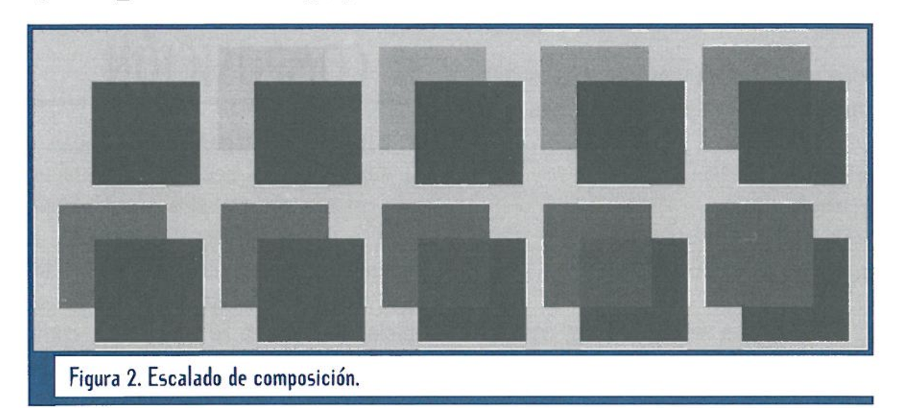


Figura 2. Escalado de composición.

Para realizar este proceso se requiere de un operador lineal denominado *kernel*, que determina unos coeficientes que aplicar sobre los *pixels* de una determinada área. Normalmente un *kernel* es una matriz cuadrada en la que el valor central es el *pixel* sobre el que se aplicara el resultado, y los colindantes los utilizados para obtenerlo. Un *kernel* clásico tiene la siguiente forma:

```
| 1/9 1/9 1/9 |
| 1/9 1/9 1/9 |
| 1/9 1/9 1/9 |
```

Por ejemplo, para aplicar el *kernel* a una determinada zona se multiplican los valores de los *pixels* que rodean al que estamos tratando por su correspondiente entrada del *kernel*. El resultado es el nuevo valor para el *pixel* central. Si el valor numérico de una zona de pixeles fuese:

```
| 6 2 9 |
| 4 7 4 |
| 3 8 2 |
```

El resultado para el *pixel* central que actualmente vale 7, aplicando el *kernel* anterior sería:

1/9*6 + 1/9*4 + 1/9*3 + ... + 1/9*9 + 1/9*2 = 5.

Este proceso se realizaría con todos los *pixels* de la imagen, por lo que este tratamiento se convierte en un proceso bastante repetitivo, algo costoso en tiempo en función del tamaño de la imagen, pero sencillo y eficaz.

El conjunto de *kernels* o matrices para realizar un determinado efecto es muy variado. Por ejemplo, mediante la siguiente matriz no se realiza ningún cambio en la imagen:

```
| 0 0 0 |
| 0 1 0 |
| 0 0 0 |
```

Pero con la siguiente, se aplica un efecto de tipo *embosse*:

```
| 1 0 0 |
| 0 0 0 |
| 0 0 -1 |
```

Otro detalle sobre los *kernels* resde en que los valores de los mismos no deberían ser mayor que uno, ya que a partir de este valor se pierde brillo en la imagen resultante. Típicamente la elección del tamaño de un *kernel* debería ser arbitraria, e incluso no restringida a matrices cuadradas, por lo que se podrían utilizar perfectamente matrices rectangulares. Según esto, el siguiente *kernel* sería perfectamente válido:

Listado 2. Ejemplo que muestra la forma de aplicar el proceso de composición.

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.applet.*;

public class e4 extends Applet {
  public void paint ( Graphics g ) {

   Image img = getImage ( getDocumentBase(),   "duke.gif");
Graphics2D g2 = (Graphics2D) g ;

g2.setBackground(Color.white);
RenderingHints qHints = new RenderingHints (
                RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON );
qHints.put ( RenderingHints.KEY_RENDERING,

RenderingHints.VALUE_RENDER_QUALITY );
  g2.setRenderingHints ( qHints );

AffineTransform at = new AffineTransform ();
  GeneralPath gp = new GeneralPath(GeneralPath.WIND_NON_ZERO);
gp.moveTo( -100.0f, -25.0f );
gp.lineTo( +100.0f, -25.0f );
gp.lineTo( -50.0f, +100.0f );
gp.lineTo( + 0.0f, -100.0f );
gp.lineTo( +50.0f, +100.0f );
gp.closePath();
g2.translate( 100.0f, 100.0f);
g2.transform ( at );
g2.setPaint( Color.blue );
g2.fill(gp);

AlphaComposite calfa = AlphaComposite.getInstance(
AlphaComposite.SRC_OVER, 0.5f);
g2.setComposite ( calfa );
g2.translate ( -30.0f, -30.0f);
g2.transform (at);
g2.drawImage ( img, at, this );
  }
}
```

```
| 1 2 3 2 1 |
| 2 4 6 4 2 |
| 3 6 9 6 3 |
| 2 4 6 4 2 |
| 1 2 3 2 1 |
```

Pero aunque este kernel representa un filtro *low-pass*, como detallaremos más adelante, en las versiones de *JDK 1.2 Beta 3* y *4* su uso produce varios errores, incluso en algunos sistemas un "cuelgue" total. Debido a esto, sólo se recomienda el uso de *kernels 3x3* hasta que se corrija esta deficiencia.

Utilizando la *API Java 2D*, realizar todo este proceso resulta bastante fácil. Se crea un objeto de tipo *java.awt.image. ConvolveOp* y se le adjunta el *kernel* que aplicar, el cual es una instancia de *java.awt. image.Kernel*. El código siguiente muestra el proceso básico de aplicación a una imagen.

```
private float[] edgeLapla = {
    0.0f, -1.0f, 0.0f,
   -1.0f, 4.0f, -1.0f,
    0.0f, -1.0f, 0.0f
  };
  miBufferOp = (BufferedImageOp)
new ConvolveOp( new Kernel(3, 3, edgeLapla));
  miBufferOp.filter(miBufferedImage, null);
```

Creamos un *kernel* con el tipo de filtro que aplicar y se lo pasamos a un objeto *ConvolveOp*. Obtenemos un objeto de tipo *BufferedImageOp* gracias al cual podemos invocar al método *filter()*, que pasándole el *BufferImage* donde tenemos nuestra imagen, aplica el objeto *ConvolveOp*, con lo que automáticamente se realiza todo el proceso de convolución a todos los *pixels* de la imagen.



Figura 3. Ejemplo de composición.

Un kernel del tipo low pass realiza un suavizado en la imagen original

Una duda que siempre aparece en el uso de *kernels* es qué ocurre con los *pixels* que son el borde de la imagen, es decir, aquellos que no tienen "vecinos" alrededor, ya sea por uno lado o por dos, por ejemplo, en el caso del *pixel* que forma una esquina. Pues bien, para estos casos la clase *ConvolveOp* incluye unas constantes que permite especificar qué tipo de valores deberán adjudicarse a estos "límites fantasmas". En un caso, podemos utilizar *EDGE_NO_OP* que especifica que los bordes de la imagen son copiados en el resultado sin modificación, es decir, no les afecta el filtro. En el caso de la constante *EDGE_ZERO_FILL* se define que los *pixels* que hacen de borde en la imagen destino tendrán un valor numérico igual a 0en el *kernel*. Por defecto, se utiliza *EDGE_ZERO_FILL* para cada objeto *ConvolveOp*.

TIPOS DE KERNELS QUE APLICAR

Existe una variedad de *kernels* que realizan diferentes acciones sobre una imagen. Los hay que suavizan los bordes de las figuras, otros que la "limpian", otros que la oscurecen, etc. Los *kernels*, como operadores lineales más utilizados, suelen ser de los siguientes tipos:

- *Low Pass.*
- *High Pass.*
- Detección de bordes.

Los *kernels de tipo Low Pass* se encargan de realizar pequeños cambios en la imagen, de forma que se obtenga una imagen más suavizada, e incluso se puede llegar a aparentar borrosa. A este tipo de filtros pertenece el conocido como *blur*. Para usar un kernel tipo *blur* se podría utilizar el código siguiente:

```
private float[] blurKernel = {
      1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f,
      1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f,
      1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f,
  };

miBufferOp = (BufferedImageOp)
new ConvolveOp( new Kernel(3, 3, blurKer-
      nel));
  miBufferOp.filter(miBufferedImage, null);
g.drawImage(miBufferedImage, 0, 0, iw, ih, this
      );
```

Aplicando el filtro a una imagen podríamos obtener el resultado que se observa en la figura 5. Como se observa, todos los bordes de la imagen aparecen difuminados, como si estuvieran borrosos. Otro tipo de filtros son los denominados de *high pass*, que en contra de los anteriores, acentúan la aparición de los colores de más frecuencia, esto es, resaltan las partes de la imagen en las que se sitúan colores muy similares y deja otras zonas menos definidas intactas.

Un kernel es una matriz cuadrada utilizada en el proceso de convolución

Debido a todo este proceso, este tipo de filtros tiene un efecto no deseado que se denomina *dithering*. Este efecto surge al tratar imágenes poco definidas y hace que en esas zonas aparezcan colores inexistentes en la imagen original debido a ese intento de resalte de zonas de mucha frecuencia, por un intento de aproximación en los colores.

El código necesario para aplicar este filtro sería muy similar al ante-



Figura 4. Foto original.

rior, únicamente habría que cambiar el *kernel* a:

```
private float[] highpass = {
    -1.0f, -1.0f, -1.0f,
    -1.0f, 9.0f, -1.0f,
    -1.0f, -1.0f, -1.0f
};
```

y usando la imagen anterior se obtendría la figura 6.

El siguiente tipo de filtros más utilizado es el de *detección de bordes*. Se suelen utilizar principalmente dos tipos de *kernels* para realizar esta tarea, ya que en función de los resultados que obtengamos en una imagen, uno u otra ofrecerá mejores resultados. El proceso de detección de bordes se basa en realizar un incremento del contraste en las zonas donde hay una diferencia mayor entre los colores. Sería como aplicar un filtro *blur*, pero potenciando el contraste de la imagen resultante.

Los filtros high pass acentúan los colores de más frecuencia

El primer tipo de kernel para esta tarea es el denominado *Prewitt Gradient*, o gradiente de *Prewitt*, que permite realzar los bordes de una imagen de una determinada dirección. Así por ejemplo, tendremos las siguientes matrices para detectar los bordes hacía la izquierda y derecha:

Izquierda:

```
| 1  1 -1 |
| 1 -2 -1 |
| 1  1 -1 |
```

Derecha:

```
|-1  1  1 |
|-1 -2  1 |
|-1  1  1 |
```

El proceso de programación sería el mismo que en los casos anteriores, solamente hay que aplicar una de estas matrices a la imagen correspondiente. Los resultados de la detección mejoran en función del tipo de imagen, por lo que en las imágenes en las que hay mayores diferencias de colores los limites estarán mejor definidos. El resultado de aplicar la matriz de izquierda a nuestra imagen ofrecería el la imagen de la figura 7.

El kernel del tipo Prewitt permite detectar los bordes en una determinada dirección

El otro tipo de *kernel* para bordes es el *kernel Laplaciano*, que permite detectar bordes en todas las direcciones. Este filtro es similar al tipo *high pass* pero hace que los valores de los colores tiendan a negro cuando se detecta un borde formado con colores muy similares, los colores de mucha frecuencia se oscurecen. La matriz típica para este filtro es la siguiente:

```
private float[] edgeLapla = {
    0.0f, -1.0f, 0.0f,
    -1.0f, 4.0f, -1.0f,
    0.0f, -1.0f, 0.0f
};
```



Figura 5. Kernel tipo Blur.

Existe una variación en las matrices de detección de bordes que se denomina *sharpening kernel* cuyo *kernel* es el siguiente:

```
| 0 -1  0 |
| -1  5 -1 |
| 0 -1  0 |
```

que puede ser aplicado por el lector y observar los resultados obtenidos. En el *CD-ROM* de la revista se incluye un programa llamado *filtros.java* en el que se pueden aplicar todos estos filtros y añadir los suyos propios a las imágenes que desea, de una forma sencilla y rápida. El programa principalmente realiza lo siguiente:

1) Lectura de la imagen que tratar.
2) Carga en una tabla de *hash* con todos los *kernels* disponibles.
3) Ejecución de los filtros en función del valor devuelto por la lista de opciones que corresponde a un tipo de *kernel* que aplicar.

Una lectura algo más detallada del código permitirá al lector modificarlo a su elección y adaptarlo a sus usos personales, comprobando la facilidad y potencia que ofrece el *API Java 2D* para realizar esta serie de tareas. Si le quedan ciertas dudas, pregunte a los programadores en C++ que realizan trabajos



Figura 6. Kernel tipo High-Pass.

sobre imágenes, ya que más de uno le hablará sobre unos inicios nada agradables. Comentar brevemente que existen operadores no lineales basados en otro tipo de operaciones sobre los *pixels* como son el *rank* y el *count filtering*.

Mediante un objeto BufferedImage se pueden realizar rotaciones, ampliaciones, reducciones, etc

Rank filtering es un ejemplo de ellos, y se basa en que de todos los colores de una zona, se elige, por ejemplo, el más oscuro o más claro según se quiera, como el resultante. Un filtro de este tipo muy conocido es el encargado de reducir el "ruido" en las imágenes, el denominado *noise filter*. *Count Filtering,* es similar al anterior, pero aparte de seleccionar el valor más alto o más bajo, usa la frecuencia de aparición para determinar la elección del mismo. Este filtro aunque también elimina el ruido de la imagen, ofrece un resultado de tipo *bold,* es decir, el resultado es una imagen que ha perdido definición y se acentúa todo mucho más.

LOOKUP TABLES

Este tipo de tratamiento de imágenes no se basa en los colores de los *pixels* de la imagen, sino en la modificación de los colores de la paleta que utiliza la misma, utilizando lo que se denomina una *lookup table* que es la estructura de datos que contiene la paleta de colores.

En este tipo de tratamiento, los colores (*RGB*) de un *pixel* son manipulados para obtener otros mediante la modificación de la paleta, en este caso la tabla. Hay que recordar que un color está formado por una composición de rojo, verde y azul (*RBG*) cada uno de los cuales contiene un valor entre **0** y **255**, siendo la mezcla de estos tres el color final.

Para realizar este tipo de modificaciones se utiliza un objeto de tipo *java.awt.image.LookupOp* y otro de tipo *java.awt.image.LookupTable*. Mediante ellos se puede separar cada una de las bandas *RBG* de un color o usar una única tabla que contiene la paleta final y manipularlas a nuestra necesidad. Por ejemplo, podríamos generar una tabla con la paleta de colores y realizar sobre ella las alteraciones que deseemos, por lo que la imagen podría pasarse a tonos de grises, o invertir colores de una forma automática.

INVERSIÓN DE COLORES

Este efecto es el producido en las fotografías, es el resultado del clásico negativo de fotos. Debido a que es un proceso de inversión de colores, si se vuelve a realizar se obtiene la imagen original, es decir, el negativo del negativo. La siguiente porción de código permite realizar este efecto:

```
short [] invierte = new short [256];
for ( cont = 0 ; cont < 256 ; cont ++ )
invierte [cont] = (short) ( 255 – cont );
miBufferOp = (BufferedImageOp)
new ShortLookupTable( 0,invierte), null);
```



Figura 7. Detección de bordes por Prewitt.

Como se aprecia, el proceso es muy parecido al utilizado en los filtros, solamente que en este caso se realiza sobre la paleta de colores. El resultado de aplicar este efecto sobre la imagen anterior es el que aparece en la figura 8.

CAMBIO EN UNA BANDA DE COLOR

Ahora bien, también se puede realizar este mismo efecto pero que únicamente afecte a un color de la composición de las bandas de *RGB*. Para ello, tenemos que tener una tabla diferente para cada color *RBG* según cada color de la paleta. Veamos como quedaría el código para realizar el cambio sobre el azul:

```
short[] invertido= new short[256];
short[] original = new short[256];
for ( i = 0; i < 256; i++) {
     invertido[i] = (short)(255 - i);
  original[i] = (short)i;
}
short[][] azul= new short[][] { original, original,
     invertido};
miBufferOp = (BufferedImageOp)
new LookupOp(new ShortLookupTable(0, azul),
     null));
```

Observando el código, realizamos el cambio en la banda azul. Para realizar-



Figura 8. Lookup table efecto de negativo.

lo en la banda del rojo, crearíamos simplemente la tabla de la siguiente forma:

```
short[][] rojo= new short[][] { invertido, original, original};
```

No hay que perder de vista que manejamos la paleta como un conjunto de tres bandas *RGB*. El resultado de aplicar este efecto se aprecia en la figura 9.

El proceso de composición define como se solapan dos figuras para generar la imagen final o mezcla

Otro efecto interesante es el denominado *Posterinzing*, que reduce el número de colores de la imagen.

```
short[] posterize = new short[256];
for ( i = 0; i < 256; i++) {
    posterize[i] = (short)(i - (i % 32));
    }
miBufferOp = (BufferedImageOp)
new LookupOp(new ShortLookupTable(0, posterize),null));
```

El efecto se basa en la aplicación del módulo del conjunto de colores que queremos utilizar, es decir, en la siguiente línea de código estamos realizando la operación módulo a todos los valores de la paleta, por lo que estamos eliminado **32** colores, como especifica la línea siguiente:

```
posterize[i] = (short)(i - (i % 32));
```

ya que si **i=255**, (i%32)=31 y por tanto i-31 = 224, que es el color máximo que se puede visualizar, luego se han eliminado los últimos **32** colores. El resultado de aplicar este efecto es el que vemos en la figura 10.

Las trasformaciones se realizan mediante una instancia de AffineTransform

Estos efectos se pueden añadir al programa anteriormente comentado sin más que insertar el resultado de cada operación en la tabla de *hash* definida y asignarle un valor en la lista de selección. Sirve para familiarizarse con el *API Java 2D* y el tratamiento de imágenes.

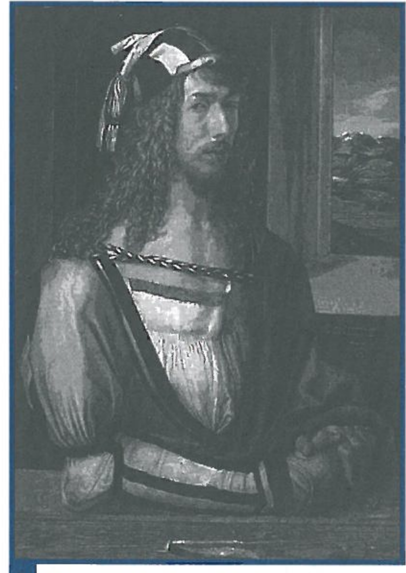


Figura 10. Lookup table, efecto Posterizing.



Figura 9. Lookup table efecto de Cambio en azul.

CONCLUSIÓN

Como se ha podido comprobar a lo largo de este artículo, el nuevo *API Java 2D* permite realizar todo un conjunto de espectaculares operaciones sobre imágenes de una forma rápida, practica y sencilla, lo que hace que sea ideal para desarrollar potentes aplicaciones relacionadas con el reconocimiento de formas, visualización, ayuda en medicina, etc. Desde las transformaciones básicas hasta el filtro más llamativo, apenas se ha tenido que escribir una centena de líneas de código, lo que demuestra la alta potencia del *API Java 2D* para todo este tipo de operaciones.

BIBLIOGRAFÍA

- "VB Graphics Programming." Rod Stephens. Wiley Computer Publishing.
- "Java Second Edition". Michael Morrison. QUE.
- Programmer's Guide to the Java 2D API. Sun.



Del HTML al acceso a bases de datos (I)

Juan Manuel Menéndez (jfrias@ibm.net)

Uno de los retos que tiene hoy día *Internet* consiste en pasar de ser un sistema en el que la información viaja principalmente desde el servidor al cliente, a ser un sistema capaz de intercambiar datos con los clientes y establecer en los servidores cierta lógica que permite el diseño de aplicaciones interactivas.

OBJETIVOS

Una de las primeras necesidades que surgió con el protocolo *HTTP* y el lenguaje diseñado para trabajar con este protocolo es que el usuario pudiera ejercer una interacción mínima con el servidor, y que esta interacción le permitiera dos tareas básicas:

- Seleccionar la información que se va a recibir.
- Enviar datos que pudieran ser procesados en el servidor.

Para conseguir estos objetivos se creó *CGI* (*Common Gateway Interface*) que como se verá más adelante permite la ejecución de unos programas en el servidor que procesan información suministrada por el cliente y devuelven una respuestas en formato *HTML*. Pronto este estándar ha mostrado sus limitaciones en el acceso a datos y procesos de transacciones conversacionales. A lo largo de este articulo y el que viene estudiaremos en detalle el estándar *CGI* mediante ejemplos prácticos con el fin de obtener una visión lo suficientemente profunda como para comprender bien las carencias que tiene para responder a las modernas necesidades de acceso a datos en *intranets/Internet*.

El estándar CGI especifica variables de entorno que se utilizan para pasar información a este tipo de aplicación

Además se verá una aproximación a las soluciones aportadas para evitar estas carencias y cuáles son las actuales tendencias en la arquitectura de procesos en este campo, que en la actualidad se encuentra en plena evolución.

Se explicará la forma de desarrollar una aplicación orientada a la red *Internet*, aunque para probar su correcto funcionamiento, se podrá utilizar una *intranet* basada en la misma plataforma. Es decir, el procedimiento para crear este tipo de aplicaciones es el mismo, al estar basado en las mismas interfaces y protocolos, de tal forma que el proceso de desarrollo resultará más asequible.

CGI

La Interfaz de Pasarela Común (*CGI*) es un protocolo estándar que establece la forma en que se deben comunicar las aplicaciones externas con los servi-

dores *Web*. Se trata de una extensión de protocolo *HTTP* que permite realizar una interacción con los servidores *Web*. Por medio de *CGI* es posible escribir programas que se ejecutarán en el servidor cuando sean invocados por el navegador *Web*. En líneas generales su funcionamiento es el siguiente:

- Se recibe un documento *HTML* con un formulario, se rellena este formulario y se pulsa el botón que envía los datos introducidos.
- El navegador mediante unos métodos específicos del protocolo *HTTP* envía los datos al servidor.
- El servidor pasa los datos a la aplicación *CGI*, que se encargará de procesarlos.
- Una vez procesados los datos la aplicación *CGI* escribe las respuestas en formato *HTML*. Esta respuesta es recibida por el servidor *Web* usando la salida estándar de la aplicación *CGI*.

El desarrollo de aplicaciones intranet requiere conocimientos del lenguaje de programación

El servidor envía la salida al navegador como si se tratara de un documento *HTML* y éste muestra el resultado o la respuesta al usuario.

En el tercer punto descrito se indica que el servidor pasa los datos a la aplicación *CGI*, acción que se efectúa mediante el uso de variables de entorno. Estas variables se establecen después de que el servidor *Web* reciba la solicitud por medio del método adecuado.

El estándar *CGI* especifica ciertas variables de entorno que se utilizan para trasladar información a una aplicación *CGI*. Se trata de las siguientes:

CONTENT_LENGTH = Tamaño de los datos enviados.

CONTENT_TYPE = Tipo de los datos.

GATEWAY_INTERFACE = Revisión de la especificación CGI que cumple el servidor.

HTTP_ACCEPT = Tipos MIME que aceptará el cliente.

PATH_INFO = Información de la ruta virtual de acceso.

QUERY_STRING = Información de la consulta.

REMOTE_ADDR = Dirección IP del equipo que hace la petición.

REQUEST_METHOD = Método que se utilizó para hacer la solicitud.

SCRIPT_NAME = Ruta virtual del programa que se ejecutará.

SERVER_NAME = Nombre del servidor, bien sea el alias DNS o la dirección IP.

SERVER_PROTOCOL = Nombre y revisión del protocolo con que se ha transportado esta solicitud.

SERVER_SOFTWARE = Nombre y versión del software servidor.

Existen otras variables que no se han indicado con el fin de no hacer una lista demasiado extensa, de tal forma que se han mencionado las más usuales. Algunas de ellas resultan bastante intuitivas, como pueden ser las que indican la longitud, el tipo de contenido o la dirección remota. Otras menos se usan para realizar aplicaciones *CGI* que sean capaces de aprovechar al máximo las características del entorno en que se están ejecutando. Así pues una aplicación *CGI* que sea capaz de aprovechar las particularidades de cada protocolo o bien explotar las opciones especiales que tenga un determinado servidor deberá investigar las variables *SERVER_PROTOCOL* y *SERVER_ SOFTWARE* respectivamente.

Tabla 1. Caracteres de control CGI.

+	Utilizado para sustituir el espacio
=	Utilizado para separar el campo nombre del campo valor
?	Indica el principio de los datos del formulario en la línea de órdenes
%	Para codificación de caracteres ASCII en hexadecimal
&	Separa parejas nombre/valor
_	Reemplaza a la barra de división

Además del conjunto de variables estándar existen otras variables que los fabricantes de los servidores añaden, por lo que siempre será necesario acudir a las especificaciones de la norma además de a las especificaciones del fabricante, aunque, y esto es una opinión particular del autor, salirse de los estándares y usar especificaciones propietarias es una medida que hay que utilizar con cuentagotas y previo convencimiento de que no existe otra vía para lograr lo que se desea obtener.

MÉTODOS GET Y POST

Los navegadores usan dos métodos para enviar los datos de un formulario *web* al servidor: *GET* y *POST*. En el método *GET* los datos se reciben en la variable de entorno *QUERY_STRING*, mientras que en el método *POST* los datos se reciben por medio de la entrada estándar.

El tipo de método se indica en el propio documento *HTML*.

En el caso del método *POST* se indicará en el documento *HTML* dentro el párrafo *form* de la siguiente manera:

```
<FORM method="POST"
ACTION=http://direccion/dir_ejec/nombre_programa>
....descripción del formulario
/FORM>
```

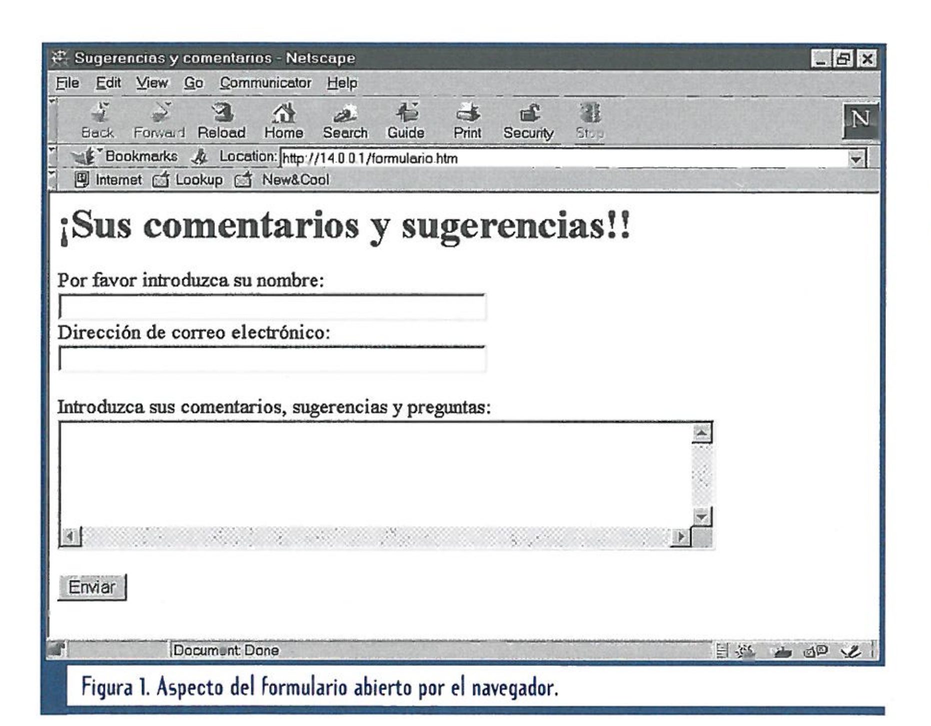


Figura 1. Aspecto del formulario abierto por el navegador.

Si se opta por el otro método únicamente es necesario sustituir la palabra reservada *POST* por la palabra *GET* asociada a *method*. El método más habitual de trabajo es *POST*, pues ciertos sistemas imponen limitaciones de espacio a la línea de parámetros con que se invoca a un programa. Además, los programas que usan la entrada estándar del sistema son de tipo consola, es decir, sin la utilización de una interfaz gráfica.

El método más habitual de trabajo es POST, otros sistemas imponen limitaciones de espacio

Conviene indicar que para crear un *script CGI* se pueden emplear diferentes lenguajes de programación, ya que como se indicó antes CGI es una interfaz. En este caso *C* será el lenguaje de desarrollo elegido, aunque los lectores podrían optar por otro. En primer lugar, un programa *C* invocado por *CGI* debe realizar lo siguiente:

- Determinar cómo ha sido llamado, comprobando el valor de la variable de entorno *REQUEST_METHOD*.
- En función de esta variable de entorno podrá determinar cómo recibirá los datos y si no está preparado para recibirlos deberá indicar la imposibilidad de aceptar dicha petición.
- Si puede aceptar la petición calculará la longitud de los datos por medio de la variable *CONTENT_LENGHT*.

Desde *C* se podrá obtener el valor de dichas variables mediante la función *getenv*.

```
char *;p
p = getenv("CONTENT_LENTGH");
```

Este valor viene en formato de texto, por lo que es necesario convertirlo en un valor entero usando la función *atoi* (*ASCII to integer*).

```
int i;
i = atoi(p) ;
```

Ya sólo queda leer, mediante la función *fgetc*, los datos que han llegado desde el cliente. Estos datos vienen en un formato especial, por lo que es necesario proceder a su interpretación para poder ser procesados.

DEL FORMULARIO AL SERVIDOR

Para poder entender mejor la representación de la cadena de caracteres vamos a volver al formulario y observaremos un formulario como el

Source of: http://14.0.0.1/formulario.htm - Netscape

```
<HEAD>
<TITLE>Sugerencias y comentarios</TITLE>
</HEAD>
<BODY>
<form action="http://14.0.0.1/cgi-bin/cgi1.exe method="POST">
<H1>¡Sus comentarios y sugerencias!!</H1>
Por favor introduzca su nombre:
<BR><INPUT TYPE=text NAME="name" SIZE = 40 MAXLENGTH=40>
<BR>Dirección de correo electrónico:
<BR><INPUT TYPE=text NAME="email" SIZE = 40 MAXLENGTH=40>
<P>
Introduzca sus comentarios, sugerencias y preguntas:<BR>
<TEXTAREA NAME=comments ROWS=5 COLS=60 MAXLENGTH=3000></TEXTAREA>
<P>
<input type="submit" value="Enviar">
</FORM>
</BODY>
</HTML>
```

Figura 2. Código HTML del documento. El formulario viene definido por la sentencia form.

que aparece en la figura 1. Se puede ver que en el documento *HTML* aparecen ventanas de texto en las que se va a introducir los datos que queremos que procese el programa. En la figura 2 se puede observar la codificación de dicho listado, en el que cada campo está definido por la sentencia *INPUT*:

```
<INPUT name=nombre_del_campo>
```

Por cada campo que tiene el formulario el navegador envía al servidor una pareja de valores con el siguiente formato:

```
nombre_campo=valor_del _campo"
```

Las distintas parejas de valores se unen entre ellas por medio del carácter **&**, de la forma:

```
nombre1=valor1&nombre2= valor2.....
```

Además de estos valores existen otros caracteres especiales que se detallan en la tabla 1. Así pues una de las cosas que es necesario realizar por el programa *CGI* es deshacer esta cadena de caracteres para obtener los valores adecuados. En los programas de ejemplo se adjuntan tres rutinas de dominio público en las que se realiza esta tarea.

Las tres funciones que realizan esta tarea son las siguientes:

strcvrt: Cambia las apariciones de un carácter por otro, en el caso de una entrada *CGI*, elimina el carácter fin de cadena por un blanco y el carácter + por el carácter blanco. Como se puede ver en la tabla 1, en origen el cliente sustituye los blancos por este signo y lo que se hace es deshacer este cambio.

UrlDecode: Busca los caracteres que vienen en forma hexadecimal, es decir precedidos por el carácter % selecciona los dos siguientes si son dígitos y llama la función *TwoHex2Int*.

TwoHex2Int: Convierte los dos caracteres hexadecimales en su valor numérico hexadecimal.

Una vez que se han puesto los datos de entrada en el formato más adecuado a nuestro programa, éste se encuentra en condiciones de realizar el proceso y devolver la salida al navegador. La salida puede ser un documento *HTML* generado dinámicamente por el programa *CGI* o puede ser un documento referenciado existente en el *Website*.

Una de las tareas que debe realizar el programa CGI consiste en deshacer la cadena de caracteres de entrada

Si el documento es generado por el programa, éste debe incluir en su salida estándar *stdout* toda la información necesaria para que el cliente entienda lo que va a recibir. En primer lugar, debe mandar al navegador información acerca del tipo de documento que va a recibir, y a continuación toda la salida como si fuera un documento-tipo *HTML*:

```
/* Manda información acerca del tipo de documento: */
printf("Content-type: text/html\n");

/* Imprime on-line el documento HTML */
printf("<HEAD> <TITLE>"Envío realizado</TITLE>("<HEAD>\n");
printf(<BODY> <h2>La información ha sido aceptada.<br>Gracias</h2>\n");
```

Si lo que se desea hacer es que el servidor envíe un documento ya existente, ubicado en cualquier parte de la Red, lo que se deberá utilizar es el código *Location*:

```
print ("Location: http://nombre.dominio.es/doc/respuesta\n");
```

INSTALACIÓN DE UNA APLICACIÓN CGI EN UN SERVIDOR WEB

Un servidor *Web* es un *software* que admite peticiones de los clientes según un estándar, procesa estas peticiones y devuelve una salida. El protocolo bajo el cual se hacen las peticiones desde los clientes a los servidores se denomina *HTTP* (*Hiper Ttext Transfer Protocol*). Sin entrar en otras consideraciones diremos que este protocolo permite la transferencia de información desde el servidor a los clientes en un modo no conectado. En el siguiente artículo se verá más detenidamente este protocolo y sus limitaciones en el diseño de aplicaciones con acceso a datos.

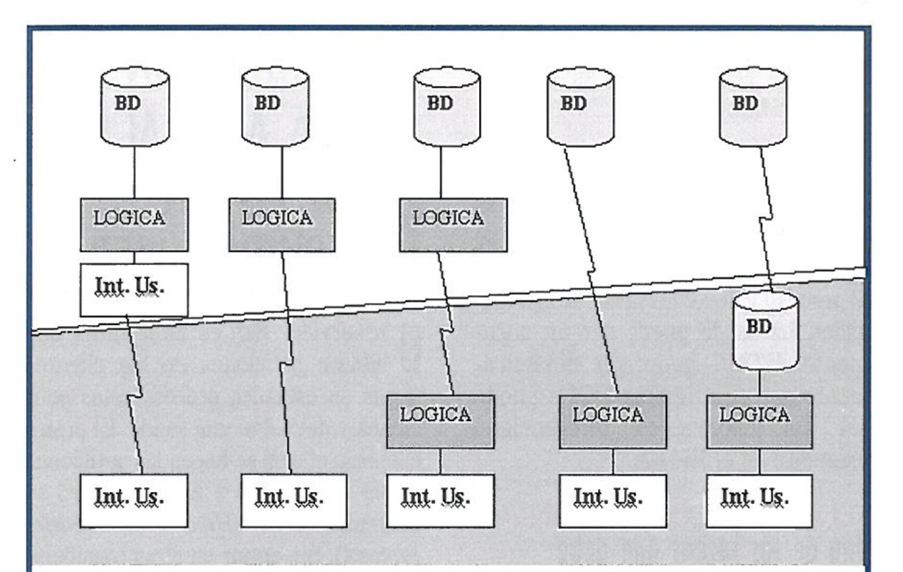
Existen en el mercado muchos y buenos servidores *Web* y a pesar de las diferencias que pueden existir entre ellos hay una parametrización común entre todos. Una de las misiones de la parametrización de los servidores es el mapeo de los nombres de los directorios virtuales con los directorios reales en el sistema anfitrión del servidor *Web*. A modo de ejemplo se verá la instalación del programa que se ha desarrollado en un servidor para sistemas *Windows NT* (*Internet Information Server*) y otro para sistemas *Unix*, como es el servidor *Apache*.

Pero antes de instalar el programa es necesario realizar una compilación. La única modificación que hay que hacer para compilarlo es comentar y quitar el comentario una sentencia de tipo *#include*, la cual deberá tener la siguiente forma bajo *Windows NT*:

```
#include <unistd.h>
```

y para Unix/Linux:

```
#include <io.h>
```



Modo 1. Presentación distribuida: Datos, Lógica y parte de la presentación son gestionadas por el servidor, parte de la presentación por el cliente, la arquitectura 3270 pertenece a este modelo.
Modo 2. Cliente gráfico: Datos y lógica residen en el servidor, los clientes sólo tienen capacidad de presentación, un ejemplo de ésta es la arquitectura Web en sus inicios.
Modo 3. Aplicación distribuida: Datos en los servidores y lógica repartida entre el servidor y el cliente, la interfaz de usuario íntegramente en el cliente. La arquitectura Web con Java es un ejemplo de este modelo.
Modo 4. Servidor de Datos: Los datos residen en los servidores y la lógica en los clientes junto con la presentación de usuario, la mayor parte de las aplicaciones pertenecen a este modelo, como es el caso de Cartera, que si bien el sistema completo es un sistema distribuido, las aplicaciones que lo componen en su mayoría funcionan en este modo.
Modo 5. Datos distribuidos: En este modelo los datos están distribuidos entre el servidor y el cliente.

Figura 1. Modelos de arquitectura Cliente/Servidor (De izquierda a derecha).

Pues la función *access* que indica si el proceso tiene acceso a un fichero se encuentra definida en el fichero referenciado en la sentencia *include*. Otra modificación de las que se deben realizar es ésta:

```
#define _MAX_PATH 256
```

Este valor corresponde al máximo valor que puede tener una cadena de caracteres que sirve para almacenar un *path* completo de un fichero no está definida con ese nombre en el sistema *Unix*. Resulta más cómodo añadirla que sustituir el nombre de este campo por el correspondiente en *Unix*. Para realizar la compilación del programa en *Unix* se ha utilizado el compilador *gcc* de la forma que se muestra a continuación:

```
gcc -o cgi1 cgi.cpp
```

Y para la compilación en sistemas *NT* se ha usado *Visual Studio* y se ha indicado para la compilación que se trata de un programa de consola. Ya compilado nuestro programa pasaremos a instalarlo en los servidores *Web* antes mencionados.

APACHE

La configuración de este servidor está incluida en cuatro ficheros que son losque nombramos a continuación: *httpd.conf*, *srm..conf*, *access.conf* y *mime.types*. La información que nos interesa aparece en el fichero *srm.conf*, concretamente en el parámetro *ScriptAlias* de la forma:

```
ScriptAlias nombre_virtual    nombre_real.
```

INTERNET INFORMATION SERVER

Este servidor de *Microsoft* se suministra junto con el sistema *Windows NT*, y aunque funcionalmente realiza la misma misión, el mecanismo de administración es completamente distinto. En el anterior servidor *Web* la configuración está basada en ficheros, a los que se accede directamente desde un editor, mientras que el acceso y configuración de este servidor se basa en un entorno gráfico.

La salida puede ser un documento HTML generado de forma dinámica por el programa CGI o puede ser un documento referenciado en el website

Para ver la configuración de este servidor desde el menú **Inicio** se arranca la aplicación *Administrador de Servicios de Internet*. Seleccionando las propiedades del servicio *WWW* se puede ver el cuadro de diálogo de *Propiedades del Servicio WWW* y dentro de la pestaña *Directorios* se comprueba que existe un directorio de ejecutables. Esto última se efectúa examinando las propiedades de cada directorio que aparece en la lista.

Con estas acciones lo único que se ha hecho ha sido colocar el ejecutable *CGI* en un directorio con permiso de ejecución, es decir, todos los ejecutables que se encuentren en estos directorios puede ser ejecutados desde un navegador mediante una petición *HTTP* lanzada desde el navegador. En cualquier caso es el método invocado el que solicita la ejecución de un programa o la descarga de un documento.

ARQUITECTURA DE PROCESOS

Los modernos sistemas de información están empezando a poner los sistemas de bases de datos en la Red y a utilizar al máximo las posibilidades de la misma para poner al alcance del público en general un tipo de aplicaciones que hace unos años sólo estaban destinadas a empresas .

Para conseguir aplicaciones de alto rendimiento se opta por diseñar la parte de lógica en C o C++

La inclusión de los sistemas de bases de datos en *intranet/Internet* ha supuesto un gran paso en la evolución de los sistemas cliente/servidor. En este caso, el modelo cliente/servidor para las aplicaciones basadas en *HTML* y *CGI* ha "aligerado" al máximo el cliente, dedicándolo exclusivamente a la parte de presentación, mientras que el resto ha pasado al servidor donde se ha realizado una división también entre la parte lógica del sistema y la parte de gestión de datos. Esta separación presenta las siguientes ventajas:

- La parte de lógica puede ser realizada en un lenguaje interpretado como *Perl*, que permite el rápido desarrollo de aplicaciones sencillas y de bajo rendimiento.
- Si por el contrario se desea aplicaciones de alto rendimiento, se puede optar por diseñar la parte de lógica en *C* ó *C++*.
- Otra cuestión que se debe tener en consideración se refiere a la portabilidad del programa. En el ejemplo que ha servido de guía se ha podido notar lo fácil que ha sido realizar la compilación para cada sistema y que prácticamente no ha sido necesario realizar modificaciones en el mismo para lograr una compilación correcta en cada sistema. Y las pequeñas modificaciones realizadas a mano se hubieran podido realizar de forma programada utilizando las directrices adecuadas del compilador para lograr un compilación condicional. En este aspecto tanto *C* (ó *C++*) como *Perl* ofrecen una portabilidad excepcional. En el punto opuesto un lenguaje como Visual Basic no ofrece portabilidad.
- Si se desea una mayor portabilidad a expensas de un rendimiento inferior se puede desarrollar la lógica en un lenguaje más universal, como es *Java*.
- La base de datos elegida como óptima para trabajar en un sistema no puede dar el mismo resultado en otro diferente.

Tabla 2. Este simple programa Perl imprime todas las variables generadas por CGI

```
#!/usr/local/bin/perl

print "Content-type: text/html\n\n";
while (($key, $val) = each %ENV) {
 print "$key = $val<BR>\n";
}
```

Todos estos apartados dan como solución ideal separar la gestión de los datos de la lógica del aplicativo, con el fin de conseguir una máxima flexibilidad. El desarrollo de aplicaciones *intranet* requiere, además de los conocimientos del lenguaje de programación, el dominio de *CGI, HTTP* y *HTML*.

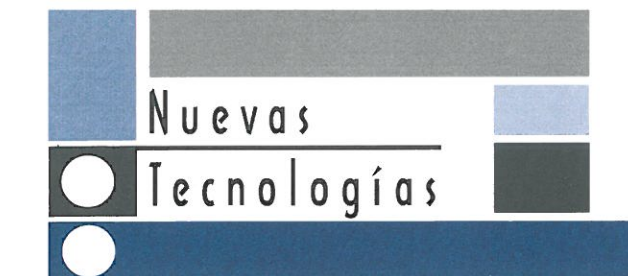
Si la aplicación *intranet* además tiene acceso a bases de datos, serán necesarios también conocimientos de sistemas gestores de bases de datos, lo que hace que para llevar a término una aplicación *intranet* sea necesario disponer de equipos de desarrolladores especializados, estando cada vez más lejos los tiempos en que los sistemas de estas características eran mantenidos, desarrollados y gestionados por pequeños equipos de personas.

DISEÑO DE UN SISTEMA DE TRES NIVELES

El sistema *CGI, HTTP* y *HTML* que utilizan las aplicaciones en *intranet/Internet* ofrecen por la propia arquitectura de este sistema una tendencia hacia la creación de sistemas de información en tres niveles.

- El primer nivel es el llamado de presentación al usuario. Este nivel es ampliamente cumplido por el estándar *HTML* en sus distintas versiones:
  - Estandariza la presentación al usuario.
  - Estandariza la codificación de la presentación.
  - Incluye todo tipo de soporte multimedia.
- El segundo nivel contiene la lógica del negocio y es donde se codifica la parte principal del aplicativo. El estándar *CGI* se encarga de la comunicación entre el primer nivel y el segundo.
- El tercer nivel contiene el almacenamiento y el acceso a los datos y lo componen los diversos sistemas de bases de datos relacionales.

En el siguiente artículo se verá un ejemplo simple de la realización de un sistema en tres niveles, a partir de los conocimientos presentados a lo largo de este artículo. Así, se desarrollarán los diferentes apartados que componen cada parte: formulario HTML, *CGI*, etc.



Novedades y cambios con Java 2

Javier Sanz Alamillo (jsanza@teleline.es)

Con la aparición de *Java 2* se presentan importantes mejoras sobre anteriores versiones, tanto en las características propias del lenguaje como en el uso de las herramientas y de la *API*, además de un amplio conjunto de nuevas posibilidades.

Java 2, nombre con el que *Sun* ha bautizado al hasta ahora conocido *JDK 1.2*, supone una nueva plataforma *Java* estable, segura y potente, actualmente disponible para entornos como *Windows NT 4.0*, *Windows 95/98* y *Solaris*, en la que se han incluido todo un conjunto de nuevas e importantes características. Ahora, por ejemplo, se dispone de una nueva *API 2D* que ofrece una sustancial mejora en el tratamiento de imágenes. Se han realizado cambios en el modelo de seguridad, aparecen las *Java Collections*, etc. Todo un amplio abanico de cambios que hacen que *Java 2* sea ideal para llevar a cabo nuevos desarrollos y portar los realizados anteriormente.

INTRODUCCIÓN

Con *Java 2* se da un paso hacia adelante en la madurez de *Java*. Las herramientas disponibles han sido mejoradas y se han incrementado sus posibilidades. Así el compilador es más estricto y genera un código más optimizado. Se han creado todo un conjunto de extensiones del *API*, las *Java Plataform Standard Extensions,* o conjunto de librerías que, sin formar parte de la *API*, ofrecen nuevas posibilidades de desarrollo. Destacan por ejemplo, *Java 3D*, que permite la construcción de aplicaciones visuales *3D* y *JCE* (*Java Criptography Extension*), en la que se ofrecen mejores posibilidades para el tratamiento seguro de la información. Surge el *Java Plug-in*, que permite que los *applets* y los *javabeans* de una *intranet* utilicen el *JRE 1.2* (*Java Runtime Extension*) en vez de la máquina virtual propia del navegador, con lo que se aprovechan todas las posibilidades de *Java 2*, lo que sin duda se convierte en una característica de gran relevancia.

Respecto a las propiedades básicas de *Java* se ha mejorado la seguridad, ciertos detalles de la persistencia y de la entrada/salida, así como un nuevo *Java Collections* que permite el manejo de estructuras de datos de una forma sencilla y rápida. Se ha creado *Java IDL*, que permite utilizar *CORBA* con *Java* de una forma simple y práctica. También se ha mejorado la velocidad en las aplicaciones que utilizan *RMI* y se han añadido nuevas clases al paquete *java.net.* Ya a nivel de *JFC* (*Java Foundation Classes*) con *Java 2D* se amplían las posibilidades gráficas, las opciones de impresión se han mejorado e incrementado y el *drag & drop* ha sido refinado.

En resumen, todo un conjunto de cambios, modificaciones, mejoras y añadidos que conforman la plataforma *Java 2*, ó *JDK 1.2* para aquellos que lo estuvieran ya utilizando, y que se detallarán a lo largo de este artículo.

Si bien se han realizado mejoras significativas, el escaso tiempo que lleva *Java 2* en el mercado hace que se generen ciertas dudas sobre su uso, aunque se requiere cierto tiempo para conocer y utilizar todas sus nuevas posibilidades. En la figura 1 se describe Java 2 con todas sus características.

CUESTIONES A TENER EN CUENTA

Los desarrolladores se preguntan si merece la pena cambiar a esta nueva versión, sin que hayan transcurrido ni

seis meses desde el último lanzamiento, o permanecer en espera a ver que pasa y luego tomar una decisión. En el aire se encuentran cuestiones como si se ha mejorado la estabilidad de la máquina virtual, si se habrán reducido y corregido esos problemas que han surgido cuando hemos programado ciertas aplicaciones y si se ha incrementado la velocidad de ejecución, de forma significativa. ¿Qué pasará con la compatibilidad con los navegadores? Según están las cosas entre *Sun* y *Microsoft*, ¿Quién garantiza que las próximas versiones de *IE* soporten *JDK 1.2*?, ¿Para cuando *JDK 1.2* para *Linux* y *Mac*?, y casi lo más importante, ¿Cuándo los grandes desarrolladores de *software*, como *Inprise* (*Borland*), *IBM* o *Symantec* ofrecerán entornos de desarrollo que soporten *Java 2*?

Se han realizado cambios en el entorno de desarrollo y de ejecución, así como en la API y en las herramientas

Todas estas cuestiones habrán sido planteadas por los programadores al tener que realizar un nuevo desarrollo con *Java*. Algunas de estas preguntas tienen respuesta aquí mismo, pero otras, sólo se podrán conocer con el tiempo.

COMPATIBILIDAD E INCOMPATIBILIDAD

Siempre que hay un cambio de versión, se presenta la duda de qué ocurrirá con todos los programas desarrollados, si seguirán funcionando o requerirán algunas modificaciones. *Sun* define claramente la compatibilidad de su nueva versión con las anteriores e incluso declara algunas de sus incompatibilidades.

COMPATIBILIDAD

Se definen dos tipos de compatibiliades que podemos denominar como compatibilidad binaria y compatibilidad de código fuente.

- **Compatibilidad binaria**

La versión 1.2 de *JDK* es compatible hacia arriba respecto a las versiones *1.0* y *1.1* excepto para las incompatibilidades que se detallarán a continuación. Esto significa, que salvo las incompatibilidades definidas, las clases creadas según las versión *1.0* y *1.1* de *Java* compilarán correctamente con la versión *1.2*.

Ahora bien, si compilamos un programa usando la versión *1.2* y la vamos a ejecutar con una máquina virtual de versión inferior, esto se denomina compatibilidad hacia abajo, observamos que aunque generalmente se cumple la compatibilidad, en algunos casos resulta problemática, por lo que *Sun* define esta circunstancia como: "hay una compatibilidad hacia abajo soportada pero no garantizada".

Debido a esto, algunos *obfuscators* (programas que impiden compilar una clase *java* para obtener el código original) al generar un fichero de clase con un formato diferente, rompen la compatibilidad hacia abajo con la versión *1.2* de *JDK*.

- **Compatibilidad de código fuente**

Java 2 es compatible hacia arriba a nivel de código fuente con versiones *1.0* y *1.1*, excepto por algunas incompatibilidades. Esto significa, que los fuentes escritos con las versiones *1.0* y *1.1* cumplen las especificaciones del lenguaje y de la *API* para *Java 2*. La compatibilidad hacia abajo no se soporta. Un programa escrito con las características de la versión *1.2* o de su *API* no podrá ser utilizado en versiones anteriores de *JDK*, como es de esperar. Los métodos *deprecated* se pueden seguir utilizando sólo por razones de compatibilidad y el compilador nos avisará con un mensaje.

Respecto al tema de los paquetes definidos bajo el paquete *sun.**, siempre hay que recordar que se trata de un producto que no debe utilizar el desarrollador y que su uso rompe todo tipo de compatibilidad e incluso puede suceder que algún programa deje de funcionar.

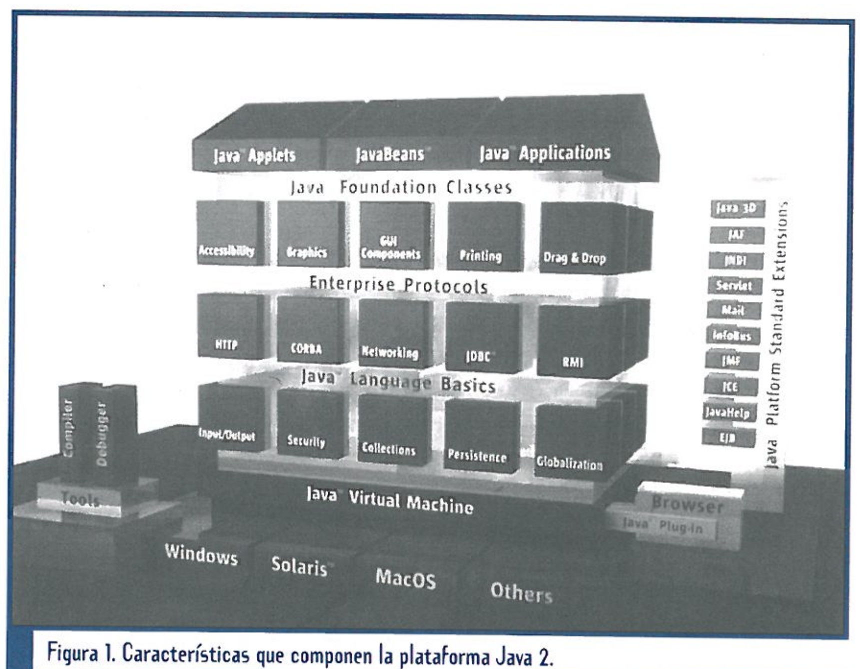


Figura 1. Características que componen la plataforma Java 2.

```
[d:\]java -version
java version "1.2"
Classic VM (build JDK-1.2-V, native threads)

[d:\]javac detecta.java
detecta.java:5: Integer literal out of range.  Decimal int literals must be in t
he range -2147483648 to 2147483647.
                int i = 3000000000;
                        ^
detecta.java:6: Integer literal out of range.  Decimal int literals must be in t
he range -2147483648 to 2147483647.
                int j = 6000000000;
                        ^
2 errors

[d:\]java -version
java version "1.1.6"

[d:\]javac detecta.java
detecta.java:5: Numeric overflow.
                int i = 3000000000;
                        ^
1 error

[d:\]
```

Figura 2. Diferencias de compilación.

INCOMPATIBILIDAD

A continuación se muestran un conjunto de casos en los que un programa escrito para las versiones *1.0* ó *1.1* de *Java* presenta algún problema o no funciona correctamente bajo la versión *1.2*. Hay tres tipos de incompatibilidades.

- **Incompatibilidades en la especificación**

A continuación se muestran las más comunes en la especificación, si bien, estos tipos de problemas no suelen aparecer en los desarrollos normales.

Java 2 consta de optimización en la localización de memoria, en el recolector, en los threads, etc.

Los compiladores de versiones *1.0* y *1.1* compilaban algunos tipos ilegales sin mostrar un *warning* o un error. Esto se ha resuelto en la versión *1.2* haciendo que el compilador sea más estricto a la hora de validar las especificaciones del lenguaje. Muchas de las incompatibilidades que a continuación se detallan son una consecuencia de seguir las especificaciones del lenguaje y no provienen de nuevos cambios. La inicialización de tipos como *long* e *int* no mostraba mensaje de aviso u error cuando se realizaban incorrectas inicializaciones.

```
public class foo {
int i = 300000000;
int j = 600000000; }
```

Las versiones anteriores a la *1.2* muestran un error en la inicialización de la variable **i**, pero nada más. La inicialización de **j** se producía incorrectamente mediante un *overflow*. Usando la versión *1.2* se elimina este error y se avisa de dos inicializaciones incorrectas. En la figura 2 se muestran las diferencias al compilar el ejemplo anterior usando *JDK 1.1.6* y *JDK 1.2*:

En versiones anteriores a la *1.2* la conversión en la asignación de un *char* a un *byte* y a un *short* se permitía en caso de que el carácter pudiera asignarse en *8 bits*. Por ejemplo: **byte b = 'b'**. En la versión *1.2* se mostrará un error, necesitando realizar un *cast* de conversión: **byte b=(byte) 'b';**

El valor **0xL** no es un valor hexadecimal válido. En versiones anteriores normalmente era utilizado como valor cero.

El *char* ''' no es válido en la versión *1.2*. Se debe utilizar '\''.

El tipo *void[]* no es válido y producirá un error de compilación.

No se admiten métodos abstractos con modificadores *private, final, native* o *synchronized*.

La asignación en determinados casos a variables "final" ya no está permitida. En las versiones *1.1.x* se daba por válido el siguiente código aunque en realidad era incorrecto:

```
int k= 10;
for (final int a ;; )
 { a=k; k++; }
```

También se permitía código del mismo estilo en versiones anteriores a *Java 2*:

```
public class prueba {
      final int k ;
      prueba () { k = 10; } }
```

No es válido hacer referencias a campos no estáticos con una expresión aparentemente "estática" del estilo **Classname.campo**. En versiones anteriores a *Java 2* la expresión era asumida como si fuera **this.campo**.

Se han corregido distintos errores en temas relativos a interfaces, *inner class*, y clases abstractas, como la invocación a métodos desde el constructor de una clase abstracta, etc. También se han ajustado algunos usos poco lícitos de *super()*.

Otra circunstancia curiosa es la siguiente. El compilador no detectaba duplicidad en la declaración de etiquetas anidadas. Por ejemplo,

```
hola: while ( condicion1 ) {
      hola: while ( condicion2 ) {
        break hola; }}
```

- **Incompatibilidades en tiempo de ejecución**

Algunas incompatibilidades de este tipo que mostraban anteriores versiones de *JDK* son las siguientes:

La máquina virtual aceptaba clases cuyos ficheros no deberían poderse utilizar de acuerdo a las especificaciones definidas. Esta circunstancia ocurría cuando el fichero de la clase contenía *bytes* añadidos al final del fichero, algunos

métodos o campos tenían un nombre que no empezaba con una letra, o desde una clase se intentaba acceder a miembros privados de otras. Estos errores se pasaban por alto en la verificación y algunos de ellos se debían al uso de *obfuscators*. La máquina virtual en la versión *1.2* es ahora muchísimo más estricta con estos problemas y con algunos menos serios pero también importantes.

Realizar algunos cambios en la variable *CLASSPATH* podía hacer que se ejecutara cierto código más privilegiado. Por ejemplo, en máquinas virtuales anteriores a la versión *1.2* se podía ejecutan ciertas clases que no tenían que estar firmadas tras pasar la verificación.

En las versiones *1.0* y *1.1* se producían ciertas circunstancias anómalas que surgían al invocar el método *finalize()*. Se puede resumir este problema indicando que cuando se disponía de una serie de objetos preparados para ser finalizados pero algunos estaban sin finalizar, eran eliminados directamente por el recolector de basura al final o en el siguiente ciclo de actuación del mismo. Por este motivo, en algunas circunstancias, como por ejemplo al utilizar *threads*, se producían *deadlocks*. Todos estos problemas han sido eliminados en la versión *1.2* y se sigue recomendando no utilizar el método *finalize() y* utilizar referencias o en casos menos problemáticos llamadas a *Runtime.runFinalization* en intervalos periódicos.

Al utilizar las clases *DeflaterOutputStream*, *InflaterInputStream*, *GZIPInputStream* y *GZIPOutputStream* se admitían constructores con valores **0** ó negativos para especificar el tamaño de *buffer*. En *Java 2* se lanza una excepción del tipo *IllegalArgumentException*. También se han corregido problemas similares en el paquete *java.io* al utilizar *null* como parámetro de entrada.

Algunas aplicaciones que han definido su propio gestor de seguridad utilizando el modelo de original no funcionarán bajo *1.2* porque hay una excepción.

Se han resuelto problemas de carga y seguridad en el uso de *ClassLoader*. Además la identificación de la versión de una clase se resuelve ahora mediante una *java.lang.UnsupportedClassVersionError* cuando anteriormente se visualizaba una excepción de tipo *NoClassDefFoundError*.

- **Incompatibilidades de la API**

Se muestran a continuación algunas de las incompatibilidades más importantes sobre la *API* de *Java*.

Los paquetes de *Swing* y de accesibilidad que anteriormente se localizaban en *com.sun.java.** han sido movidos a *javax.**. Las aplicaciones antiguas deberán cambiar en los programas el nombre del paquete para ejecutarse correctamente. Para realizar esa tarea, existe una herramienta denominada *PackageRenamer*, que facilita todo este proceso.

La API ha sido ampliada y se han añadido nuevas posibilidades, como el tratamiento de imágenes y sonido

La interfaz *java.util.List* ha sido añadido a *Java 2*. Como ya existe una clase *List* en otro paquete, en *java.awt.**, si realizásemos la importación de los dos paquetes, el compilador mostraría un error. Para solucionarlo, debemos utilizar *import java.awt.List* ó realizar un nombrado global cuando se utilice un objeto de tipo *List*.

Se ha cambiado la *signature* (conjunto formado por el nombre de una función, sus parámetros y el tipo de excepción que puede lanzar) del método *java.io.StringReader.ready* para permitir que se controlen *IOException*. Los problemas al usar el método *decode()* con tipos *Integer* y *Short* han sido corregidos.

Ciertas circunstancias sobre tipos *HashTable* y *Vector* han sido eliminadas, como por ejemplo, la de incluir como clave en una *HashTable* una referencia de sí misma, lo cual anteriormente provocaba un *StackOverflow*. La clase *java.io.File* ha sido definida de forma más clara, solucionando ciertos problemas como la construcción del *String* que representa la localización de un fichero en disco y la eliminación de separadores sobrantes para localizar un fichero. El paquete *java.sql* ha crecido por la creación de los métodos *Connection*, *DatabaseMetaData*, *ResultSetMetaData*, *ResultSet*, *CallableStatement*, *PreparedStament*, *Stament*. Como se ha comprobado, se han solucionado muchos problemas, por lo que el resultado es una implementación *Java* mejorada y mucho más estable.

CAMBIOS Y NUEVAS CARACTERÍSTICAS

Dado que el conjunto de cambios que ofrece *Java 2* es bastante amplio, éstos se pueden agrupar en cuatro categorías: cambios en el entorno de desarrollo y de ejecución, cambios en las herramientas de desarrollo, mejoras en la *API* y ampliación de esta última.

CAMBIOS EN EL ENTORNO DE DESARROLLO Y DE EJECUCIÓN

Dentro de este tipo de cambios podemos englobar los relativos al manejo de la variable de entorno *CLASSPATH*, las mejoras en el *JIT* (*Just In Time*), así como las modificaciones en los *threads*, tanto en métodos *deprecated* como en nuevas implementaciones.

- **Cambios en el classpath**

Aunque este tipo de cambio también podría incluirse en la parte de

herramientas de desarrollo, lo trataremos aquí por estar directamente unido a la ejecución de los programas. En la versión *1.1*, el *class path* se determinaba bien mediante un valor por defecto y un valor obtenido de la variable de entorno *CLASSPATH*, o mediante la opción -*classpath* de ejecución. Se seguía el siguiente orden:

- *Class path* por defecto: define el valor del directorio de instalación de *JDK*, del estilo **c:\jdk1.1.6** en *Windows*, es decir, dónde se encuentra el fichero *classes.zip* o *rt.jar* tras la instalación.
- Variable de entorno *CLASSPATH*, determinada por el usuario, que contendría a su vez el *class path* por defecto, añadiendo generalmente el directorio actual mediante ".".
- Opción de ejecución *–classpath*, que indica dónde encontrar los ficheros *classes.zip* o *rt.jar* y las clases de nuestras aplicaciones.

La última opción no suele ser muy utilizada ya incluir el directorio de *classes.zip* se producen muchos errores, incluso uno se llega a olvidar con el tiempo de dónde está instalado.

Usando *Java 2* este problema se ha terminado, ya que no hace falta indicar el directorio de clases de *JDK*, ya que son buscadas por defecto y únicamente hay que añadir los directorios que deseemos utilizar, como si usásemos una variable de entorno *CLASSPATH*. Podríamos poner por ejemplo:

```
java –classpath d:\misclases prog "hola"
```

Si por algún motivo se cambia el directorio de instalación, no se podría utilizar este sistema, aunque sí funciona con la versión *1.1* El orden de búsqueda de las clases con Java 2 es el que indicamos a continuación:

1.- Se buscan las clases en el directorio que indica la propiedad *bootstrap classpath*, que es asignada automáticamente con el directorio de ejecución o directorio que contiene el fichero *rt.jar*. Se puede averiguar el valor mediante *System.getProperty ("sun.boot.class.path")*. Por ejemplo:

```
C:\Program Files\JavaSoft\JRE\1.2\lib\rt.jar;C:
    \Program
Files\JavaSoft\JRE\1.2\lib\i18n.jar;C:\Program
    Files\JavaSoft\JRE\1.2\classes
```

2.- A continuación se busca en los directorios indicados por el valor de la propiedad *("java.ext.dirs")*, que puede tener un valor como el siguiente:

```
C:\Program Files\JavaSoft\JRE\1.2\lib\ext
```

3.- Por último se buscan las clases en el *classpath* de la aplicación, indicado por *("java.class.path")*, obteniéndose ".", que es el directorio actual.

Extensiones Framework

La pregunta siguiente es: ¿y cómo indicamos el lugar donde se sitúan otras clases?. Al eliminar el uso de la variable de entorno *CLASSPATH* se han definido una serie de directorios donde las situaremos. Los ficheros *jar* se deben colocar a partir del directorio *...\lib\ext* del *runtime* de *Java*, especificado por *...\jre\lib\ext*. Si lo que se tiene son unas clases, se sitúan a partir del directorio *\jre\classes*. Así de simple.

Se han mejorado las herramientas de desarrollo como Javadoc y Jar

Appletviewer ignora el uso de la variable *CLASSPATH*, ¿cómo indicamos nosotros en un *applet* dónde tiene que buscar las clases?. La solución es bastante sencilla:

1.- Las clases se buscan en los directorios indicados por el navegador.
2.- Las clases se buscan en función de la opción *codebase* del *applet*.

Mejoras en la velocidad

Con *Java 2* se espera conseguir velocidades de ejecución muy próximas a las obtenidas por los programas escritos en C++ y sobre todo si se usa *HotSpot JVM*, según información de *JavaSoft*. De todas formas, *JDK 1.2* incluye unos cambios que mejoran la velocidad de ejecución. Se trata de:

- Localización de memoria y recolección de basura más rápida, gracias al uso de *heaps* o montículos locales para cada *thread*, con lo que se pueden eliminar los bloqueos para localizar memoria dinámica. Gracias a esto, a su vez el recolector de basura es menos utilizado y por menos tiempo, es por lo que se reducen las pausas en el programa cuando éste es ejecutado.
- El consumo de memoria se ha reducido gracias a que se pueden compartir las cadenas constantes de diferentes objetos.
- Las librerías nativas como por ejemplo *AWT* han sido reescritas usando *JNI* (*Java Native Interface*).
- Los monitores de los *threads* son muchos más rápidos, por lo que los métodos *synchronized* se ejecutan casi tan rápido como los métodos normales.
- Mejora de rendimiento en el *RMI*. Por la parte cliente, se mejora la rapidez gracias a cambios realizados en el recolector de basura distribuido (*DGC*) que elimina cuellos de botella en la sincronización general de las operaciones. En la parte servidor, se utiliza un nuevo sistema de referencias a objetos y notificaciones que eliminan la búsqueda de los objetos que generar. La carga dinámica de clases también ha sido optimizada y se ha mejorado bastante la velocidad en la serialización de las clases *RMI*.

Java 2 incluye un *JIT*, mediante el que se aumenta la velocidad de ejecución. Por defecto, está activado. Para desactivarlo y poder así realizar una traza de los programas, se debería poner a **NONE** la propiedad *"java.compiler"*. La opción *–nojit* no tiene ningún uso.

- **Threads "nativos" para Solaris**

En las versiones anteriores, *Java* no hacía uso de los distintos procesadores instalados en máquinas *Sun* con *Solaris*. Ahora gracias a la inclusión de soporte nativo para *threads*, se puede usar el modelo original o *green* (aún por defecto) o el nuevo y nativo. Para ello, se ejecuta el programa añadiendo el *flag –native* en la línea de comandos. El soporte nativo mejora el uso de los *threads* y posibilita su uso en paralelo si es posible, a la vez que se evitan ineficaces llamadas de *I/O*.

- **Métodos "deprecated" de los threads**

Varios métodos de las clases *Thread* y *ThreadGroup* han sido *"deprecated"*. Esto significa que no deben ser utilizados, ya que tarde o temprano se eliminarán de *Java*. Esto debería haber ocurrido desde la versión *1.0* del *JDK*, pero inicialmente eran necesarios. Los métodos que se deben evitar de las dos clases anteriores son *stop()*, *suspend()* y *resume()*. Ocurre lo mismo con el método *countStackFrames()* de la clase *Thread* y *allowThreadSuspension()* de *ThreadGroup*. Los tres métodos anteriores deben ser evitados porque no son seguros. Suelen dejar en un estado "no adecuado" a los *threads* sobre los que actúan o producir interbloqueos.

CAMBIOS EN LAS HERRAMIENTAS DE DESARROLLO

A continuación explicamos los principales cambios en estas herramientas. La utilidad *jar* se ha ampliado con dos nuevas opciones. Una de ellas permite añadir o eliminar archivos de un fichero *jar* sin tener que eliminarlo y volverlo a crear. Así, mediante la opción *–u* se añade un fichero nuevo al *jar* ya existente.

```
jar uf mijar.jar nuevofich.class
```

Mediante la opción *–C* se pueden modificar los directorios de donde se crean o actualizan los ficheros que pertenecen a un fichero *jar*. Las opción *–O* de *javac* en *Java 2* no implica la activación de la opción *–depend*, por lo que debe ser activada de forma manual. Además, el efecto sobre la optimización no es el mismo. Usando *javap –verify* en anteriores versiones se realizaba una mínima verificación sobre la clase. En *Java 2* esta opción ha sido eliminada.

Siguiendo esta línea, en anteriores versiones de *JDK* se permitía introducir desde la línea de comandos redundancia y combinaciones de parámetros, como por ejemplo, varios parámetros *–classpath*. Este problema ha sido eliminado.

Usando Java 2 se consigue mantener la compatibilidad hacia arriba

La herramienta *javadoc* utiliza un nuevo sistema de nombrado:

- La documentación resultante se genera en un directorio en función de la jerarquía creada en vez de un simple directorio. Los nombres de los paquetes indican nombres de directorios y no de ficheros como antes.
- Se genera una pagina inicial *index.html* basada en *frames* de *HTML*.

Debido a errores en las versiones *1.1.X* de *JDK*, el código firmado mediante el uso de *javakey* no será reconocido por *Java 2*, al igual que el código firmado mediante *JDK 1.2* no será reconocido por versiones anteriores. En *Java 2*, la herramienta *javakey* ha sido eliminada. Ahora se realiza la misma tarea utilizando las herramientas *keytool* y *jarsigner*. Además, existe una tercera herramienta, *policytools* que permite utilizar las nuevas opciones de control de acceso de *Java 2*.

MEJORAS EN LA API

Se han realizado todo un conjunto de mejoras en el *API* que lo hacen más consistente, estable y fácil de utilizar. Se trata de las siguientes:

- **Cambios en las Clase File**

Los programadores comprobarán las grandes mejoras en el uso de esta clase, ya que si usando *JDK 1.1* la mayoría de los métodos devolvían un objeto tipo *String* para representar un fichero, ahora también se puede obtener un objeto *File* con el que poder trabajar. Por ejemplo, si tenemos que calcular la suma del tamaño de todos los ficheros de un directorio en *Java 2* haríamos:

```
File mifich = new File ( .... );
long tamTotal = 0;
File tabfich [] = mifich.listFiles();
int totfich = tabfich.length;
for ( int cont=0 ; cont < totfich ; cont ++ )
tamTotal += tabfich[i].length();
```

Mientras que con versiones de *JDK* anteriores hacemos:

```
File mifich = new File ( .... );
long tamTotal = 0;
String tabfich [] = mifich.list();
int totfich = tabfich.length;
for ( int cont=0 ; cont < totfich ; cont ++ )
tamTotal += new File (
        mifich,tabfich[i]).length();
```

Se pueden generar ficheros temporales mediante *createTempFile()*. Se puede convertir un objeto tipo *File* en tipo *URL mediante* el método *toURL()*, comprobar si un fichero es de tipo oculto con *isHidden()* o poner como de sólo lectura mediante *setReadOnly()*.

- **Mejoras en el audio**

Se ha modificado el *engine* de sonido existente, de forma que ahora sí se pueden reproducir ficheros de sonido de tipo *WAV, AIFF, AU, MIDI* o *RMF* con una gran calidad. Aunque sin estar incluido en la *API* formalmente, puesto que se desarrolla con el nombre de *Java Sound*, pronto estará integrado. Ya no

Figura 3. Aplicación de sonido.

se necesitará utilizar el problemático paquete *sun.audio* o crear objetos *AppletContext* para poner sonido a nuestros *applets*. Utilizando el nuevo método estático *newAudioClip()* de *java.applet.Applet*, se pueden crear fácilmente *AudioClips* tanto para *Applets* como para aplicaciones.

Con estas mejoras se pueden crear programas que reproduzcan por ejemplo ficheros *MIDI*, permitiendo que suenen por distintos altavoces, controlar el volumen, etc. Una aplicación de ese estilo tendría una interfaz similar al que se muestra en la figura 3.

- **Cambios en tipos numéricos**

Se han añadido dos nuevos métodos, *Float.parseFloat* y *Double.parseDouble* con tipos numéricos elementales. Por ejemplo, ahora se puede realizar:

```
float f = Float.parseFloat ("3.14");
double d = Double.parseDouble ("3.14");
```

Los cálculos en coma flotante han sido mejorados y se han corregido algunos tipos de desbordamientos.

- **Mejoras en seguridad**

Las mejoras de seguridad *en Java 2* han sido notables, de tal forma que ahora surgen los conceptos de "permisos" y "política". Cuando se carga el código, se le asignan unos permisos basados en unas determinadas condiciones, en una política de seguridad. Cada permiso especifica qué tipo de acciones puede realizar, qué recursos puede utilizar (cómo leer, escribir, conectar, etc.. El tipo de política a utilizar puede ser el aplicable desde un fichero externo de configuración. Gracias a ello, se pueden detallar con mayor precisión las condiciones de seguridad, por lo que éstas pueden ser más flexibles, configurables y ampliables. Este tipo de control ya no es exclusivo para los *applets*, ya que se puede extender a aplicaciones e incluso a *javabeans*.

Se han incluido servicios de criptografía que mejoran los disponibles en el *JDK 1.1* como *MessageDigest*, *Signature* y *KeyPairGenerator*. Se está desarrollando una nueva librería de seguridad, el *Java Cryptography Extension* (JCE) que ampliará sustancialmente todas las posibilidades de *Java* en temas de criptografía. Debido a las restricciones de exportación de E.E.U.U. no forma parte de *JDK 1.2*.

La compatibilidad de código fuente no se cumple si se utiliza las nuevas posibilidades de la API

- **JDBC 2.0**

Desde su aparición en *JavaOne'98*, la *API* de *JDBC 2.0* amplía las posibilidades del *JDBC*. La mejora más importante es la posibilidad de crear auténticos "cursores" en los accesos a los datos, permitiendo desplazarse hacia arriba, abajo, realizar modificaciones sobre el resultado, etc. También se pueden utilizar los tipos de *SQL3* y soporte del uso de *blobs* (*Binary Large Object*), *clobs(Character Large Objetcs)* y tipos definidos por el usuario (*UDTs*). Se han desarrollado nuevos *drivers* de *JDBC* adaptándolos a *Java 2*. Los anteriores seguirán funcionando pero sin poder utilizar las nuevas posibilidades.

- **El paquete java.net**

Se han introducido mejoras en las clases como *URL*, que ahora permite el acceso con clave a determinadas direcciones *web*. En el paquete *java.net* se han incluido nuevas clases, tales como son:

- *JarURLConnection*, clase que permite acceder a un fichero *jar* mediante una *URL*.
- La clase *URLDecoder*, que complementa la existente *URLEncoder*.
- *URLClassLoader*, que permite la carga de clases desde una *URL* en vez de un fichero.

- **Mejoras en RMI**

Se puede realizar persistencia sobre objetos remotos mediante *Remote Object Activaction* y se pueden utilizar mediante *SSL* (*Security Socket Layer*).

- **Serialización y control de versiones**

Se pueden serializar los campos de forma explícita usando un *array* gracias a la nueva clase *serialPersistentFields*. Con los métodos *writeReplace()* y *readResolve()* se puede tener una acceso a bajo nivel en la serialización.

Se han definido nuevos *tags* para *javadoc*, como son *@serial*, *@serialField*, *@serialData* para los documentos que contengan clases de serialización.

Es posible realizar un control de versiones a nivel de paquete de forma que los *applets* y las aplicaciones se pueden identificar en tiempo de ejecución mediante una versión de *JRE*, de la *VM* y del paquete.

- **Cambios en el AWT**

La modificación más interesante reside en el soporte de internacionalización mediante las clase *ComponentOrientantion*, pero además se ha añadido un nuevo constructor a *GridBag-*

Constrainst para facilitar el uso del gestor de composición *GridBagLayout* y el nuevo método *setState()* permite iconificar y desiconificar un componente de tipo *Frame*. Se pueden utilizar todas las *fonts* disponibles por el sistema o al menos, los disponibles utilizando *Toolkit* y sacando la lista correspondiente, mediante *GraphicsEnvironment*. Por ejemplo:

```
GraphicsEnvironment ge =
new GraphicsEnvironment.getLocal Graphic-
        sEnvironment();
String lista [] = ge.getAvaliableFontFamilyNa-
        mes();
Font mifont = new Font ( lista[1], Font.PLAIN,
        12);
```

- **JavaBeans**

Aunque *Java 2* no incluye *Enterprise JavaBeans(EJB)*, se ha generado un nuevo paquete de clases, *java.beans.beancontext*, dentro de lo que se denomina el *Runtime Containment and Services Protocol*, gracias al cual un *bean* puede incluirse en otro.

- **Drag & Drop**

Con esta característica se pueden mover datos de aplicaciones "nativas" a aplicaciones *Java*, entre aplicaciones *Java* y en una misma aplicación *Java*.

- **Reflection**

Con el uso de esta clase se controla el acceso de forma explícita a un campo, método o constructor de un objeto.

AMPLIACIÓN DE LA API

Las modificaciones en este sentido son numerosas , ya que se han realizado muchos cambios en las librerías y clases, además de añadirse nuevas posibilidades. Esto ha producido que el número de paquetes *standard* haya pasado de los **23** de *JDK 1.1* a los **70** de *Java 2*.

- **Collections y Sorting**

Con *Java 2* se abre todo un conjunto de posibilidades para el manejo de estructuras de datos. Se pueden utilizar *arrays*, vectores y tablas de *hash* entre otros. Ya no es necesario utilizar la librería de *ObjectSpace's*, la *Generic Collection Library* para *Java*. Se disponen de unas **25** clases que permiten trabajar de forma rápida y sencilla con la mayoría de las estructuras de datos sin importar su implementación interna, reduciendo el esfuerzo de desarrollo e incrementando su velocidad de ejecución.

También se ha incluido la posibilidad de ordenar datos: *Array.sort* (mitabla). El único requisito es que se debe implementar la interfaz *Comparable*. La ordenación se realiza con el método natural, como si los objetos fueran texto. Para utilizarlo, se puede implementar la interfaz anterior en nuestra la clase o definir una clase aparte en la que se especifica cómo realizar la ordenación.

Tanto AWT como RMI han sido mejorados de una manera considerable

- **Referencias a objetos**

Se amplían posibilidades sobre la gestión de las referencias a los objetos o *weak references*, con el objetivo de mejorar la velocidad de ejecución y reducir la utilización del recolector de basura. Del mismo modo se ha creado una clase, *WeakHashMap*, que permite realizar una gestión sobre la vida de algunos de los objetos.

- **Java IDL**

Se ha incluido la posibilidad de utilizar *CORBA*, con un soporte de interoperación entre *CORBA* y *JRE* a través de *Java IDL* (*Interface Definition Language*). Para usar *CORBA* en los programas, se debe crear un fichero *IDL* que defina las interfaces utilizadas para la comunicación, similar al sistema utilizado en *RMI*. Tras ejecutar *idltojava* se obtienen un conjunto de clases, lo que se denominan *stub* y un *skeleton*, a través de las que se realiza la comunicación.

- **Java 2D**

Se trata de un conjunto de clases para el tratamiento de imágenes y gráficos. Se pueden visualizar textos, imágenes, líneas, etc. de una forma sencilla. La *API* ofrece un gran conjunto de posibilidades: composición de imágenes, tratamiento del canal *alfa*, gestión de la paleta de colores y un sistema de coordenadas, así como un conjunto de operadores para el tratamiento de imágenes. Todas las nuevas clases se han incluido en los paquetes *java.awt* y *java.awt.image.*

- **Input Method Framework**

Gracias a este soporte se pueden introducir caracteres *multibyte* (como los que son utilizados en el japonés, chino o coreano) en los componentes de texto del *AWT*. Actualmente, *TextArea* y *TextField* no soportaban esta posibilidad porque utilizaban las características del sistema para su funcionamiento.

- **Imprimir con Java 2**

En *Java 2* se posibilita la automatización de trabajos de impresión. Mediante el paquete *java.awt.print,* la interfaz *Printable,* la clase *PrinterJob* y el método *print()* ahora se puede realizar esa tarea. Se definen constantes como *Printable.PAGE_EXISTS* y *Printable.NO_SUCH_PAGE.*para realizar el proceso de forma sencilla.

CONCLUSIÓN

Java 2 representa una mejora bastante considerable con respecto a las versiones anteriores, ya que se han realizado muchos cambios, que no afectarán a los programas ya desarrollados. Además, se ha mejorado la estabilidad y la rapidez de la máquina virtual, también destaca la facilidad y sencillez a la hora de utilizar Java para programar.

Desarrollo de aplicaciones con videoconferencia (I)

Constantino Sánchez Ballesteros (constantino@nexo.es)

En la presente serie de artículos aprenderemos los aspectos más importantes de la programación de audio y vídeo sobre la Red de redes (*Internet*). Se tratarán las áreas destacadas para compartir programas y recursos y crear videoconferencia mediante la programación bajo *C/C++*, *Visual Basic* y *NetMeeting*.

INTRODUCCIÓN

Desde la aparición de *Internet*, a nivel de usuario siempre se deseó comunicarse con otras personas de la forma más natural posible. Al principio se utilizaban los *IRC's* (*chats*) que aún siguen en la brecha, donde los usuarios se envían mensajes escritos.

Desde que *Microsoft* creó *Netmeeting* el tema cambió. Implementó funcionalidades para poder realizar videoconferencia (esto es la combinación de imagen y sonido) entre varios usuarios, en diferentes puntos, y a través de *Internet*.

Además, con el precio de las tarifas telefónicas, se podría al mismo tiempo hablar y ver a una persona en Japón o Nueva York por el módico precio de una llamada local.

Además, se podían intercambiar ficheros y datos de forma instantánea para que todo fuera interactivo. Actualmente, y con la velocidad que ofrece *Internet* no se obtiene un rendimiento demasiado bueno a la hora de transmitir audio y vídeo, pero es posible mantener conferencias relativamente buenas.

El SDK de NetMeeting permite utilizar librerías para crear fácilmente programas de videoconferencia

Existen otros programas que permiten crear videoconferencia, pero nos centraremos en *Netmeeting*, ya que su *SDK* puede hacer que casi cualquier programador consiga crear sus propias y exclusivas aplicaciones de videoconferencia y diseñarlas a su gusto, de una manera muy entretenida y sencilla.

NETMEETING

El *SDK* de *Microsoft NetMeeting 2.1* consta de una serie de *API's* que permiten a los programadores integrar capacidades de conferencia en sus programas utilizando *Visual Basic* o *C/C++*.

Por ejemplo, se pueden realizar proyectos como los que enumeramos a continuación:

- Capacidad para intercambio de datos financieros.

- Programa orientado a médicos para colaborar en una operación quirúrgica.
- Videoconferencia en general.

El *SDK* de *NetMeeting* incluye un control *ActiveX* para conferencia que permite a los creadores de contenido *Web* utilizarlo con soluciones de *Script* para *ActiveX* (como *JavaScript* y *Visual Basic Scripting Edition*) y agregar directamente funcionalidades de conferencia a las páginas *Web*. También puede ser utilizado para crear aplicaciones *software* destinadas a desarrolladores, profesionales de páginas *Web*, etc. A continuación se detallan algunas de estas posibles soluciones de cara al mercado profesional:

- Integración de capacidades de conferencia/datos multipunto.
- Desarrollo de soluciones de conferencia para ofrecer servicios diversos.
- Funcionalidad de conferencia integrada dentro de páginas *Web*.

La programación se realizará bajo el modelo COM de C y controles ActiveX para Visual Basic

Para programar y utilizar las características que ofrece *NetMeeting* tenemos a nuestra disposición el siguiente soporte:

- Una interfaz *COM* que permite a los desarrolladores agregar la funcionalidad de *NetMeeting* en sus aplicaciones, controlando llamadas, sustituyendo la interfaz de usuario que tiene por defecto, y creando aplicaciones que trabajen en una conferencia basada en esta *API* de *Microsoft*.
- Una *API* cliente para el acceso completo al protocolo de directorios (*LDAP*) para poder acceder a las características del directorio del servidor local de Internet (*ILS*) utilizado por *NetMeeting*.
- Instalación de *Codecs* para permitir a los creadores de *Codecs* audio/vídeo la instalación de sus productos y utilizarlos en las llamadas de *NetMeeting*.

DEPENDENCIAS

Para poder ejecutar aplicaciones que utilicen el *SDK* de *NetMeeting 2.0* o una versión superior debe estar instalada en nuestro ordenador. Conviene advertir que sólo se ejecuta bajo *Windows 95/NT 4.0* o versiones superiores.

Si queremos utilizar esta *API* bajo *Windows NT 4.0* necesitamos instalar el *Service Pack 3* y el *driver* para compartir recursos. Si queremos utilizar el control *ActiveX Conference Control* también será necesario tener instalado en nuestro ordenador el programa *NetMeeting*.

Como opinión personal debo decir que *Microsoft* podría haber impuesto un poco de independencia a la hora de programar bajo esta *API*, ya que es un poco engorrosa la necesidad imperiosa de tener instalado en nuestro equipo el programa *NetMeeting* cuando los programadores sólo necesitaremos la *API* para crear nuestras soluciones de videoconferencia.

ESTÁNDARES E INTEROPERABILIDAD

Netmeeting tiene soporte para algunos de los estándares más utilizados en las comunicaciones que utilizan audio/vídeo. Estos estándares permiten que productos *hardware* de diferentes empresas se comuniquen y entiendan a la perfección cuando establecemos conexiones con nuestros equipos. De este modo, cada usuario puede tener un modelo diferente de cámara de vídeo, tarjeta de sonido, etc., pudiendo establecer comunicaciones sin ningún tipo de problema. Entre los estándares que soporta *NetMeeting* caben citar:

- *ITU H.323*: utilizado para videoconferencia y audio.
- *ITU T.120*: para gestión de datos en conferencias multipunto.
- *IETF LDAP*: para la utilización del directorio de servicios que albergará a los usuarios que quieren unirse a una conferencia.

EL ESTÁNDAR H.323

H.323 es un estándar que especifica cómo se comunican los equipos y servicios multimedia sobre redes que no permiten una calidad de servicio garantizada (como es el caso de *Internet*). *H.323* puede gestionar vídeo en tiempo real, voz y datos, o cualquier combinación de estos elementos. Los productos que utilicen *H.323* para audio y vídeo permiten conectar y comunicar con otras personas que estén en *Internet*, al igual que otras personas utilizan diferentes modelos de teléfonos para comunicarse.

H.323 posee una serie de estándares que definen los siguientes puntos:

- Cómo contestar llamadas.
- Capacidad de negociación (proceso para que dos equipos se comuniquen el uno con el otro).
- Cómo se transmiten los datos por la red.
- *Codecs* de audio y vídeo (codifican y decodifican las entradas y salidas de audio y vídeo para la comunicación entre nodos).

Para su utilización con poco ancho de banda el estándar *H.323* tiene *codecs* para audio (*G.723*) y vídeo (*H.263*) que permiten a los programas que utilicen el *H.323* el envío y recepción de voz e

Tabla 3. Componentes que gestionan la transmisión de datos.	
T.120 data	Envía y recibe datos entre diferentes participantes de la conferencia utilizando el protocolo T.120.
Construcción Datos aplicación	Usa el componente de datos T.120 para enviar y recibir información.

ILS

El componente *Internet Locator Server* (*ILS*) se utiliza para obtener usuarios que estén en un directorio determinado dentro de la Red.

DETECCIÓN DE CODECS

A continuación vamos a desglosar las partes más importantes del primer ejemplo en lenguaje *C* que utilizará los recursos y librerías de *NetMeeting* para detectar, agregar, ordenar y eliminar los *codecs* que tengamos instalados en nuestro equipo.

Una vez creado el ejecutable, debemos copiarlo al mismo directorio donde resida el ejecutable de *NetMeeting* (*CONF.EXE*) para que nuestro ejemplo pueda extraer correctamente dichos *codecs*.

Los codecs determinan la velocidad y calidad a la que se transmitirán los datos

También será necesario tener configurado el entorno de *Visual C* o el compilador de C utilizado por el lector para que sepa dónde buscar las librerías y *headers* del *SDK* de *NetMeeting*.

```
#include <windows.h>
#include <windowsx.h>
#include <objbase.h>
#include <initguid.h>

#include "resource.h"
#include "codecs.h"

typedef enum {AUDIOCODEC, NONE}
    CODECTYPE;
```

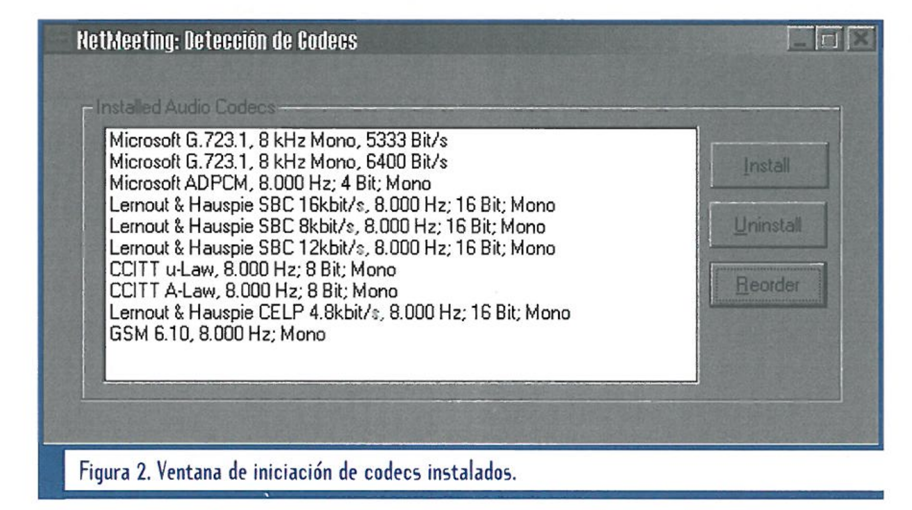


Figura 2. Ventana de iniciación de codecs instalados.

Definimos unas variables globales:

```
const int MID_SZ = 128;
// tipo de Codec que será visualizado, eliminado
    u ordenado.
CODECTYPE gtCodec = AUDIOCODEC;
HINSTANCE ghInstance;
```

Creamos las estructuras utilizadas para agregar o eliminar *codecs*:

```
WAVEFORMATEX *gpWFmt;
AUDCAP_INFO gWInfo;
```

La siguiente variable se utilizará para albergar una lista de los *codecs* de audio instalados y sus propiedades:

```
AUDCAP_INFO_LIST *gpsAudCapInfoList =
    NULL;
```

Esta variable de tipo entero contiene, en el orden de instalación, índices de las estructuras en la variable anterior **AUDCAP_INFO_LIST**:

```
int *gpnAudCapOrderedIndex = NULL;
//******************
//Interfaces Globales
//******************
static IInstallAudioCodecs *pAudio=NULL;
```

Definimos las funciones principales de la interfaz de usuario para poder ordenar *codecs*, eliminarlos, salir del programa, etc.

```
LRESULT CALLBACK MainUIProc (HWND
    hwnd, UINT message,
UINT wParam, LONG lParam);
LRESULT CALLBACK ReorderUIProc (HWND
    hwnd, UINT message,
UINT wParam, LONG lParam);
BOOL InitAppInstance (HINSTANCE hInstance);
void DeinitAppInstance(HINSTANCE hInstance);
//********************
//Funciones misceláneas
//********************
```

En las siguientes funciones se realizarán las tareas:

- Cargar los *codecs* instalados.
- Indexar los *codecs* instalados.

- Actualizar la lista a partir del índice.
- Visualizar los *codecs* instalados.

```
void LoadInstalledCodecs(void);
void IndexInstalledCodecs(void);
void UpdateCapInfoListsFromIndex(void);
void DlgDisplayInstalledCodecs(HWND);
```

La siguiente es una función de ayuda para convertir el tipo HRESULT en un mensaje de error:

```
TCHAR gszHRError[MID_SZ];
inline void HRErrorToString(HRESULT hr)
{
FormatMessage(FORMAT_MESSAGE_FROM_
    SYSTEM,NULL,
(DWORD)hr,LANG_SYSTEM_DEFAULT,gszH-
    RError,200,NULL);
}
```

Creamos la función de inicialización (*WinMain*) para la posterior creación de nuestra ventana. *Windows* reconoce esta función por el nombre como el punto de entrada inicial al programa.

Cuando se visualizan por primera vez los codecs se ordenan automáticamente por el orden

Esta función llama a la rutina de inicialización de la aplicación.

```
int PASCAL WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpszCmdLine, int nCmdShow) {
    BOOL fProcessed=InitAppInstance(hIns-
    tance);
    DeinitAppInstance(hInstance);
    return fProcessed;
}
```

La siguiente función inicializa la instancia de la aplicación:

```
BOOL InitAppInstance (HINSTANCE hInstan-
    ce) {
    WNDCLASS wc;

    ghInstance=hInstance;
```

Reordenación de Codecs por prioridad de uso

Installed Codecs:

Microsoft G.723.1, 8 kHz Mono, 5333 Bit/s
Microsoft G.723.1, 8 kHz Mono, 6400 Bit/s
Microsoft ADPCM, 8.000 Hz; 4 Bit; Mono
Lernout & Hauspie SBC 16kbit/s, 8.000 Hz; 16 Bit; Mono
Lernout & Hauspie SBC 8kbit/s, 8.000 Hz; 16 Bit; Mono
Lernout & Hauspie SBC 12kbit/s, 8.000 Hz; 16 Bit; Mono
CCITT u-Law, 8.000 Hz; 8 Bit; Mono
CCITT A-Law, 8.000 Hz; 8 Bit; Mono
Lernout & Hauspie CELP 4.8kbit/s, 8.000 Hz; 16 Bit; Mono
GSM 6.10, 8.000 Hz; Mono

Ok | Cancel | Up | Down

Figura 3. Esta ventana muestra los codecs preparados para su reordenación.

Registramos *WNDCLASS* para la aplicación. Esta clase activa la caja de diálogo principal que contiene propiedades tales como el icono de la aplicación:

```
wc.style = CS_DBLCLKS | CS_SAVEBITS
    | CS_BYTEALIGNWINDOW;
wc.lpfnWndProc = DefDlgProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = DLGWINDOWEXTRA;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon(hInstance, MAKEIN-
    TRESOURCE(IDI_INSTCODC));
wc.hCursor = LoadCursor(NULL,
    IDC_ARROW);
wc.hbrBackground = (HBRUSH)GetStoc-
    kObject(WHITE_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = "CodecInstClass";

if (!RegisterClass(&wc))
    return FALSE;
```

Activamos las estructuras de los *codecs* de audio. En concreto, vamos a activar el *codec* **GSM 6.10, 8,000 Hz, Mono.**

```
    BYTE gWFmtEx[2] = {0x40,0x01};

gpWFmt = (WAVEFORMATEX *)
    LocalAlloc(LPTR,sizeof(WAVEFORMA-
    TEX)+sizeof(gWFmtEx));
```

En la estructura del *codec* debemos definir diferentes aspectos tales como los canales, tamaño, *samples* por segundo, etc.

```
gpWFmt->wFormatTag = 49;
gpWFmt->nChannels = 1;
gpWFmt->nSamplesPerSec = 8000;
gpWFmt->nAvgBytesPerSec = 1625;
gpWFmt->nBlockAlign = 65;
gpWFmt->wBitsPerSample = 0;
gpWFmt->cbSize = sizeof(gWFmtEx);

CopyMemory((gpWFmt + 1),&gWFm-
tEx,gpWFmt->cbSize);


ZeroMemory(&gWInfo,sizeof(AUDCAP_I
NFO));
gWInfo.wFormatTag = 49;
gWInfo.uAvgBitrate = 13000;
```

```
gWInfo.wCPUUtilizationEncode = 50;
//arbitrary
gWInfo.wCPUUtilizationDecode = 20;
//artitrary
gWInfo.bSendEnabled = 1;
gWInfo.bRecvEnabled = 1;
```

Activamos el soporte *COM*. La aplicación será un diálogo de tipo modal:

```
if (FAILED(CoInitialize(NULL)))
  return FALSE;

return
DialogBox(hInstance,MAKEINTRESOURCE(ID
D_MAINUI), 0,(DLGPROC)MainUIProc);
}
```

La siguiente función inicializa la instancia de la aplicación:

```
void DeinitAppInstance(HINSTANCE hInstance)
{

//fin del soporte COM
CoUninitialize();

//liberar la memoria reservada
LocalFree (gpWFmt);
}
```

MainUI_OnInitDialog es llamada cuando la ventana principal se inicializa. Activa los punteros iniciales y actualiza la ventana de la forma correspondiente:

```
BOOL MainUI_OnInitDialog(HWND hwnd,
      HWND hwndFocus,
LPARAM lParam)
{
      HRESULT hr;
      TCHAR szString[MID_SZ];

      gtCodec=NONE;
```

Obtenemos una interfaz a los métodos de los *codecs* de audio. Para ello utilizaremos *COM* mediante *CoCreateInstance*:

```
hr = CoCreateInstance(CLSID_InstallCo-
decs,      NULL,
CLSCTX_INPROC_SERVER,
IID_IInstallAudioCodecs,
            (void **) &pAudio);

if (FAILED(hr)) {
 pAudio=NULL;
 HRErrorToString(hr);
 LoadString(ghInstance,IDS_GETIAU-
DIO,szString,MID_SZ);
 MessageBox(NULL,gszHRError,szS-
tring,MB_OK);
}
else {
 gtCodec = AUDIOCODEC;
}
```

En caso de no haber disponible ninguna interfaz desactivaremos todo:

```
if (gtCodec == NONE)  {

EnableWindow(GetDlgItem(hwnd,IDC_I
NSTALL),FALSE);

EnableWindow(GetDlgItem(hwnd,IDC_U
NINSTALL),FALSE);

EnableWindow(GetDlgItem(hwnd,IDC_R
EORDER),FALSE);

EnableMenuItem(GetMenu(hwnd),0,MF_
BYPOSITION|MF_GRAYED);
 return TRUE;
}
```

Cargamos los formatos de los *codecs* instalados:

```
LoadInstalledCodecs();
```

Visualizamos los *codecs* en la ventana de diálogo del programa (contenida en la variable **hwnd**):

```
DlgDisplayInstalledCodecs(hwnd);
return TRUE;
}
```

Cuando se termine la aplicación, destruiremos todas las estructuras, interfaces, etc., utilizadas en las inicializaciones de objetos:

```
void MainUI_OnDestroy(HWND hwnd)
{
```

Liberamos las estructuras *CapInfo*.

```
if (gpsAudCapInfoList)
 pAudio->FreeBuffer(gpsAudCapInfo-
List);
```

Vaciamos las interfaces.

```
if (pAudio)
 pAudio->Release();
```

Liberamos índices contenidos en *CapInfoIndexs*.

```
if (gpnAudCapOrderedIndex)
 LocalFree(gpnAudCapOrderedIndex);
}
```

La siguiente función se crea para comenzar todas las rutinas de instalación, desinstalación y ordenación de *codecs* de audio para su posterior visualización dentro de la ventana del programa.

```
void MainUI_OnCommand(HWND hwnd, int id,
      HWND hwndCtl, UINT codeNotify) {
      TCHAR szString[MID_SZ];
      TCHAR szString2[MID_SZ];

      switch(id)
      {
```

Instalamos el *codec* **GSM** descrito anteriormente, para lo que utilizaremos el método *AddACMFormat*:

```
case IDC_INSTALL:
 {
       if (gtCodec==AUDIOCODEC)
 {
   if (FAILED(pAudio->AddACMFor-
mat(gpWFmt,&gWInfo))) {

LoadString(ghInstance,IDS_IAFMT,szS-
tring,MID_SZ);

LoadString(ghInstance,IDS_AAFMT,szS-
tring2,MID_SZ);

MessageBox(hwnd,szString,szString2,M
B_OK|MB_ICONEXCLAMATION);
       }
 }
```

```
        LoadInstalledCodecs();
        break;
    }
```

En caso de que el usuario decida desinstalar el *codec* se ejecutará la siguiente rutina, basada en la utilización del método *RemoveACMFormat*:

```
case IDC_UNINSTALL:
 {
          if (gtCodec==AUDIOCODEC)
 {
                    if (FAILED(pAudio->RemoveACMFormat(gpWFmt))) {

LoadString(ghInstance,IDS_UAFMT,szString,MID_SZ);

LoadString(ghInstance,IDS_RAFMT,szString2,MID_SZ);

MessageBox(hwnd,szString,szString2,MB_OK|MB_ICONEXCLAMATION);
          }
 }

          LoadInstalledCodecs();
          break;
}
```

Si el usuario pulsa el botón de ordenar crearemos una nueva ventana de diálogo para efectuar esta tarea:

```
case IDC_REORDER:
 {
DialogBox(ghInstance,MAKEINTRESOURCE(IDD_REORDER),
hwnd,(DLGPROC)ReorderUIProc);
  break;
  }
}

//Visualizamos codecs instalados para su posterior reordenación

  DlgDisplayInstalledCodecs(hwnd);
}
```

La siguiente función (*LoadInstalledCodecs(void)*) carga una lista de los *codecs* instalados con sus propiedades correspondientes. Es la parte más importante del programa:

```
void LoadInstalledCodecs(void) {
```

Eliminamos cualquier información existente mediante la liberación del *buffer* (método *FreeBuffer*):

```
if (gtCodec==AUDIOCODEC) {
 if (gpsAudCapInfoList)
          pAudio->FreeBuffer(gpsAudCapInfoList);
```

Para obtener los *codecs* de audio instalados utilizaremos el método *EnumFormats*:

```
  if (FAILED(pAudio->EnumFormats(&gpsAudCapInfoList)))
          gpsAudCapInfoList=NULL;
  else
  {
  if (gpnAudCapOrderedIndex)
  LocalFree(gpnAudCapOrderedIndex);
  gpnAudCapOrderedIndex = (int *)LocalAlloc(LPTR,
sizeof(int) * gpsAudCapInfoList->cFormats);
  }
  }

  IndexInstalledCodecs();
}
```

NOTA: las estructuras devueltas por EnumFormats no garantizan que estén en orden

IndexInstalledCodecs actualiza el puntero **CapOrderedIndexes** para que contenga listas de los índices de *codecs* representados en **CapInfoLists** en el orden de su instalación.

```
void IndexInstalledCodecs(void) {
        unsigned int i;

        if (gtCodec==AUDIOCODEC &&
        gpsAudCapInfoList)
          for (i=0;i<gpsAudCapInfoList->cFormats;i++)
            gpnAudCapOrderedIndex[gpsAudCapInfoList->aFormats[i].wSortIndex]=i;
}
```

La siguiente función actualiza **CapInfoLists** mediante el orden instalado representado en la variable **CapOrderedIndexes:**

```
void UpdateCapInfoListsFromIndex(void) {
        unsigned int i;

        if (gtCodec==AUDIOCODEC)
          for (i=0;i<gpsAudCapInfoList->cFormats;i++)
            gpsAudCapInfoList->aFormats[gpnAudCapOrderedIndex[i]].wSortIndex=i;

}
```

Para finalizar con el repaso de las funciones, sólo nos queda la encargada de visualizar los *codecs* instalados en el orden especificado por **CapOrderedIndexes.**

Para enviar datos a la ventana, utilizaremos la variable **hwnd** que representa el formulario:

```
void DlgDisplayInstalledCodecs(HWND hwnd) {
        unsigned int i;
```

Visualizamos los codecs instalados que son:

```
        HWND hListBox =
        GetDlgItem(hwnd,IDC_LBINSTALLEDCODECS);
        SendMessage(hListBox,LB_RESETCONTENT,0,0);

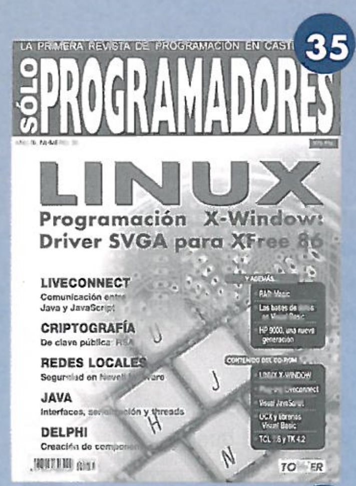
        if ((gtCodec==AUDIOCODEC)&&
        gpsAudCapInfoList)
          for (i=0;i<gpsAudCapInfoList->cFormats;i++)
            SendMessage(hListBox,LB_ADDSTRING,0, (LPARAM)(LPCSTR)
gpsAudCapInfoList->aFormats[gpnAudCapOrderedIndex[i]].szFormat);
}
```

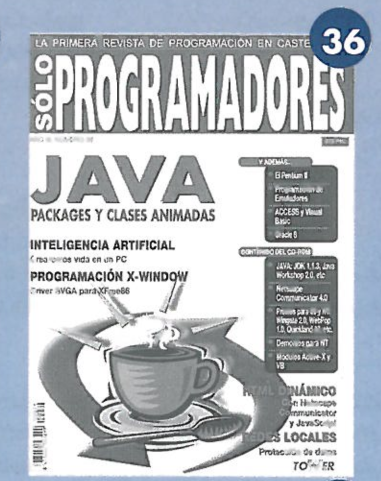
En la siguiente entrega de la serie aprenderemos a utilizar capacidades de conferencia mediante la incrustación de controles en páginas *Web* y el uso de *scripts*.

números atrasados

SÓLO PROGRAMADORES

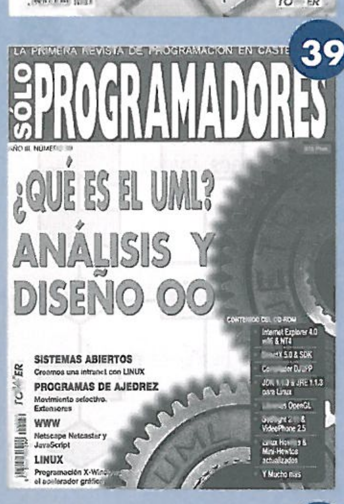
completa ya tu colección



















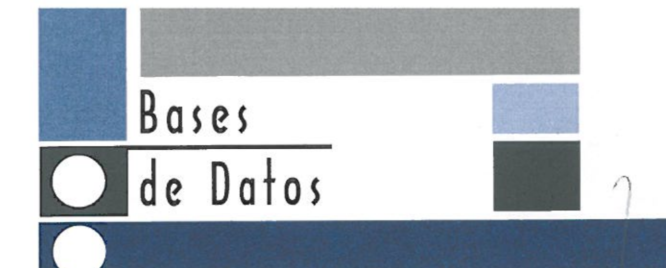












Acceso a Oracle 8 desde C++ Builder (I)

Rafael Corchuelo y Victoria Rus (corchu@lsi.us.es)

Oracle 8 es hoy en día uno de los sistemas de gestión de bases de datos más potentes y robustos que existen en el mercado. Además esta versión está abierta al desarrollo de aplicaciones distribuidas en *Internet*.

INTRODUCCIÓN

Oracle 8 es un sistema de gestión de bases de datos potente, robusto y estable que está marcando los estándares de la industria informática en este tema. Por otra parte, *C++ Builder* es una herramienta visual de desarrollo rápido de aplicaciones bajo *Windows* con la que podemos obtener muy buenos resultados en un tiempo muy inferior al de su inmediata competidora: *Microsoft Visual C++*. Además, dado que el lenguaje base de *C++ Builder* es uno de los más utilizados podemos incorporar fácilmente en nuestras aplicaciones bibliotecas de utilidad que podremos encontrar en *Internet*.

Oracle 8 incorpora un entorno de desarrollo de aplicaciones muy potente en el que podemos utilizar desde el tradicional *PL/SQL* hasta *Java*. El problema es que si una empresa ha estado utilizando *C++* para desarrollar aplicaciones, el coste de formación de personal que supone utilizar esas herramientas específicas de desarrollo no es razonable. Por suerte, *C++ Builder* ofrece unos mecanismos muy simples para acceder a las características de *Oracle 8* de forma que podamos aprovechar toda la potencia que proporciona este gestor de bases de datos al tiempo que hacemos uso de todas las características de *C++ Builder* para el desarrollo rápido de aplicaciones.

Las nuevas características de Oracle 8 permiten integrar nuestras aplicaciones en Internet

En este pequeña serie presentaremos algunas de las mejoras más importantes que incorpora *Oracle* 8, la forma de utilizar el explorador de *C++ Builder* para crear o consultar bases de datos y los componentes fundamentales para acceder a las mismas desde nuestras aplicaciones.

NUEVAS CARACTERÍSTICAS DE ORACLE 8

En su versión *8, Oracle* ha dado el gran salto hacia la interconectividad total. Es la última generación de uno de los sistemas de gestión de bases de datos líder en el mercado y es el primero que ha sido desarrollado íntegramente con la intención de aprovechar las grandes posibilidades que proporciona *Internet* a las empresas. En este sentido, una de sus mayores aportaciones es la posibilidad de almacenar e incorporar información multimedia compuesta por imágenes, audio o vídeo en la base de datos de forma que se pueda recuperar e incorporar con facilidad en aplicaciones escritas

en *Java* que se ejecutan dentro de un navegador. Además, se han incluido nuevas características que permiten aumentar enormemente el rendimiento de las bases de datos.

ACCESO A INTERNET

No cabe la menor duda de que *Java* es el lenguaje de *Internet* y que cada día son más las empresas que se deciden por él a la hora de desarrollar sus productos. De hecho, una buena parte de las utilidades que proporciona *Oracle* se encuentran escritas en este lenguaje. *Oracle 8* incorpora en el motor de bases de datos una versión de la máquina virtual *Java* especialmente diseñada para ser robusta y eficiente. Además ofrece la posibilidad de compilar *Java* a código de la máquina virtual, portable entre distintas plataformas pero lento, o código máquina nativo, menos portable pero mucho más rápido.

PARTICIÓN DE TABLAS

Supongamos que estamos trabajando con una tabla que contiene un histórico de operaciones realizadas en las cuentas de una entidad bancaria. Los campos de esta tabla podrían ser los siguientes:

Preparar nuestro equipo para acceder a Oracle es bastante simple gracias a Oracle Installer

Es evidente que a lo largo de un período de tiempo lo suficientemente largo, esta tabla puede llegar a almacenar un volumen considerable de información. Lo importante aquí es que aunque todos los datos que mantiene son necesarios, en raras ocasiones es preciso acceder a las operaciones realizadas en dos cuentas a la vez. Lo normal es que cada consulta se lleve a cabo sobre una única cuenta.

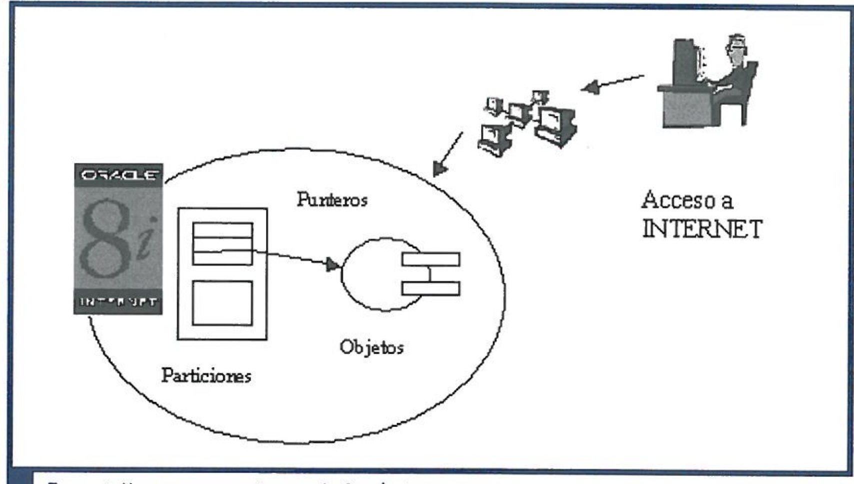


Figura 1. Nuevas características de Oracle 8.

Para hacer más eficiente la recuperación y el almacenamiento de la información, *Oracle 8* permite dividir las tablas horizontalmente en varias porciones que engloban a todos los registros que cumplen una determinada propiedad. Al hacer esto, cada porción en que resulta dividida la tabla se almacena por separado de forma que las consultas y actualizaciones de una porción no involucran al resto de las porciones que pueden estar sufriendo tratamientos similares en paralelo.

En nuestro ejemplo anterior, una forma bastante adecuada de partir la tabla podría ser utilizando el campo *COD_CTA* de forma que dentro que cada porción quedasen todos aquellos registros relacionados con la misma cuenta bancaria.

OBJETOS

Oracle 8 es el primer sistema de gestión de bases de datos relacionales que se ha decidido a incorporar extensiones orientadas a objetos que permiten introducir en las tablas, además de los tipos de datos habituales, tipos de datos definidos por el usuario. Para estos se ha elegido el paradigma de orientación a objetos, por lo que es posible definir clases a partir de las cuales se pueden obtener objetos que se pueden almacenar, recuperar y manipular dentro de las tablas de la base de datos utilizando extensiones al lenguaje *SQL*. Por desgracia, no es posible utilizar el mecanismo de herencia para definir unas clases de objetos en función de otras.

Además de las ventajas que la introducción del paradigma de orientación a objetos suponen de por sí, los objetos permiten aumentar en gran medida el rendimiento de las consultas gracias a que con ellos es posible introducir en las tablas de las bases de datos punteros a través de los cuales se puede hacer referencia directa a otros objetos, sin necesidad de utilizar consultas.

Por ejemplo, si en nuestra tabla de operaciones bancarias tenemos un registro con los datos **(015, '10/10/98', 027, 1000)**, la única forma de saber cuál es la cuenta **015** y la operación **027** es realizando una consulta de la forma:

Tabla 1. Histórico de operaciones.

HISTÓRICO			
COD_CTA	FECHA	COD_OPER	IMPORTE

Tabla 2. Estructura de la tabla con las operaciones realizadas en una cuenta.

CUENTAS					
COD_CTA	SALDO	...	HISTÓRICO		
			FECHA	COD_OPER	IMPORTE

```
SELECT *
    FROM CUENTAS, OPERACIONES
    WHERE COD_CTA = 015 AND
    COD_OPER = 027
```

Aunque definiendo índices adecuados es posible ejecutar esta consulta de una forma muy rápida, implica poner en marcha toda la maquinaria de *Oracle* para interpretar la instrucción *SQL*, consultar los índices, optimizar el acceso a los mismos, recolectar estadísticas que ayudarán a mejorar el rendimiento de consultas futuras, etcétera. Sería mucho mejor que en la tabla *HISTÓRICO* las columnas *COD_CTA* y *COD_OPER* hiciesen referencia directamente a los objetos que describen esas cuentas y esas operaciones.

Este concepto de puntero es habitual en los lenguajes de programación, pero no había sido incorporado hasta el momento en los sistemas de gestión de bases de datos. Su influencia en el rendimiento es enorme puesto que permite acelerar en gran medida operaciones tan frecuentes como la consulta de descripciones o detalles.

TABLAS ANIDADAS

Otra de las nuevas posibilidades que ofrece *Oracle 8* es la creación de tablas cuyas columnas contienen nuevas tablas de datos. Esta nueva posibilidad permite almacenar información relacionada de una forma muy eficiente y fácil de manejar.

En el ejemplo que estamos tratando, hemos utilizado una tabla histórica para almacenar las operaciones que se realizan sobre una cuenta bancaria, pero otra posibilidad mucho más compacta es introducir la información histórica como un campo dentro de la tabla que tiene los datos sobre cada una de las cuentas. En el siguiente esquema se muestra la estructura que debería tener la tabla, donde *HISTÓRICO* es un campo que realmente contiene una tabla con todas las operaciones realizadas sobre una determinada cuenta.

PREPARANDO NUESTRO EQUIPO PARA ACCEDER A ORACLE 8

Antes de poder acceder a las bases de datos, debemos preparar nuestro equipo para poder conectarnos con un servidor en el que esté instalado *Oracle*. La forma más simple de hacerlo es insertando el *CD-ROM* titulado *Oracle8 XXX Edition*, donde *XXX* es la versión de *Oracle* que tenemos (*Lite*, *Enterprise*, etc.), y seleccionando la opción **Begin Installation.**

Oracle Net8 Easy Config permite comprobar las conexiones con el servidor Oracle de una manera sencilla

Esto nos conduce a una pantalla en la que debemos dar el nombre de nuestra empresa y podemos seleccionar un directorio para instalar la aplicación y el lenguaje en el que queremos que funcione. Esta pantalla no tendría mayor interés si no fuera por un pequeño problema que existe al seleccionar el lenguaje, que debe de ser uno de los soportados por el servidor que estamos utilizando. Generalmente, el servidor tan sólo suele soportar **Inglés**, pero el programa de instalación selecciona por defecto el lenguaje en el que está funcionando *Windows*, **Español** en la mayoría de los casos. Si realizamos la instalación utilizando la versión española todo irá bien hasta el momento en que intentemos conectar con el servidor. En ese momento, se producirá un error casi imposible de descifrar y mal documentado en el manual de instalación.

Una vez que hayamos elegido un directorio adecuado y un lenguaje que sea soportado por nuestro servidor, podemos continuar adelante e instalar el programa *Oracle8 Client*, que dará acceso a todas las posibilidades que ofrece el servidor *Oracle* y además instalará todos los *drivers* y bibliotecas de enlace dinámico necesarias para poder conectar nuestras aplicaciones en *C++ Builder* con el servidor.

Si todo ha ido bien en unos minutos terminará la instalación y tendremos en el menú de inicio un grupo de programas con todas las utilidades *Oracle*. Lo primero que debemos hacer es comprobar que en efecto podemos establecer una conexión con nuestro servidor y para ello podemos utilizar el programa **Oracle Net8 Easy Config.** Cuando lo arrancamos aparece una pantalla en la que se nos advierte que se han detectado comentarios en uno de los ficheros de configuración y que estos pueden resultar afectados al utilizar este programa. . No hay que preocuparse si tarda un poco en arrancar, ya que se trata de una utilidad escrita en *Java* y es bastante lenta en máquinas que no cuentan con microprocesadores de última generación.

A continuación elegiremos la opción para crear un nuevo "servicio".

En la terminología de *Oracle*, un servicio no es más que un nombre simbólico o alias que permite hacer referencia a una base de datos. Por defecto, *Oracle* crea un usuario llamado **SCOTT** y una base de datos llamada **ORC1**. Puede ser un buen ejemplo para intentar conectarnos. Por lo tanto, lo que haremos será crear un servicio al que vamos a llamar **SCOTT.ORC1** (cualquier otro nombre sería correcto, pero es habitual nombrar los servicios utilizando el nombre del usuario y la base de datos a la que acceden separados por un punto). La figura 2 muestra la pantalla en que se efectúa esta operación.

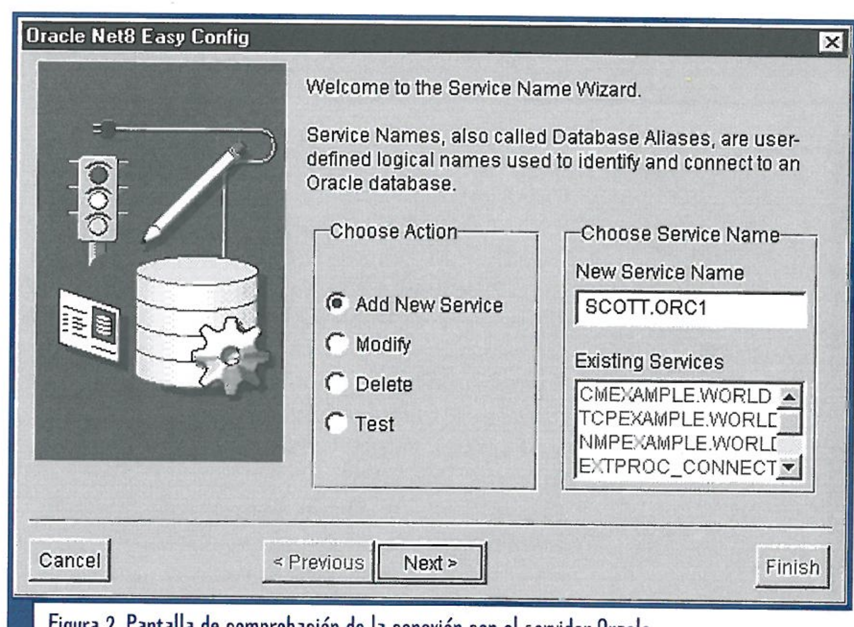


Figura 2. Pantalla de comprobación de la conexión con el servidor Oracle.

Pulsaremos el botón **next** y a continuación el programa preguntará por el protocolo de acceso a la base de datos. Lo normal si tenemos nuestro servidor en una máquina *Unix* será elegir el protocolo TCP/IP, pero también podemos usar *SPX/IPX* para acceso a través de redes *Novell*, *Named Pipes* para acceso a redes *OS/2* ó *IPC* si el servidor está en la misma máquina. En nuestro caso elegiremos *TCP/IP* puesto que nuestro servidor reside en una máquina *Sun*.

Al crear una tabla debemos tener cuidado de nombrarla a ella y a sus campos en mayúsculas

Al presionar de nuevo sobre el botón **next** pasaremos a una pantalla en la que tendremos que elegir el nombre de la máquina en la que está el servidor *Oracle* y el puerto de comunicaciones que está utilizando. Por ejemplo, **murillo.fie.us.es** en el puerto **1521**. Si hemos instalado *Oracle* con las opciones por defecto, esté será el puerto seleccionado. En caso contrario deberemos consultar al administrador de la base de datos cuál es el puerto de comunicaciones que está usando *Oracle*.

Al pulsar el botón **next** pasamos a una nueva pantalla en la que se pregunta a qué *SID* queremos conectarnos. En la terminología *Oracle*, un *SID* es el nombre que el administrador le ha dado a la base de datos a la que nos queremos conectar. Por defecto, esté será **ORC1**.

Si volvemos a pulsar **next** pasaremos a una nueva pantalla en la que, pulsando el botón **test,** podremos comprobar si es posible establecer una conexión con el servidor. Al pulsarlo deberemos introducir el nombre de usuario y la clave con la que vamos a entrar en el servidor. El nombre del usuario de prueba es **scott**, la clave deberá pedírsela a su administrador.

Si ha seguido los pasos anteriores con cuidado y ha seleccionado en la instalación un lenguaje que esté soportado por su servidor, la prueba tendrá éxito y habrá configurado su equipo correctamente para poder acceder al servidor *Oracle* desde *C++ Builder*. En caso de que se haya producido algún error en la conexión, se le informará con un mensaje que suele ser bastante específico y fácil de entender, salvo en caso de que se haya equivocado en la selección del lenguaje.

Instalar *C++ Builder* de forma que pueda acceder a *Oracle* es muy simple. Tan sólo debe recordar que durante el proceso de instalación debe indicarle que instale un *SQL Link* para *Oracle*.

CONFIGURACIÓN DEL DRIVER PARA ORACLE DE C++ BUILDER

Una vez instalados *Oracle 8 Client* y *C++ Builder* con soporte para *Oracle*, debemos configurar el *driver* de acceso mediante el programa *BDE Administrator* que se encuentra en el grupo de programas de *C++ Builder*. Al ejecutar este programa aparece una pantalla en la que debemos seleccionar la pestaña **Configuration/Drivers /Native/Oracle.**

Los dos parámetros más importantes del *driver* de *Oracle* son:

- **DLL32:** En este campo debemos indicar la *DLL* de acceso para *32 bits*. Para *Oracle 8* será **SQLORA8.DLL** y para *Oracle 7* será

Tabla 3. Protocolos de acceso a los servidores Oracle.

Nombre	Significado
3270	Protocolo específico de IBM
APPC	Protocolo IBM APPC LU versión 6.2
DECNET	Protocolo de acceso a redes de Digital
Named Pipes	Protocolo de acceso a redes de OS/2
NETBIOS	El popular protocolo usado por LAN Manager, por ejemplo.
SPX/IPX	El protocolo de Novell Netware
TCP/IP	El protocolo más habitual para intercambio de información en Internet. El habitual si nuestro servidor Oracle está instalado en una máquina UNIX
VINES	Protocolo Banyan VINES
TNS	Transparent Network Substrate. Es un protocolo específico de Oracle pensado para optimizar la velocidad de transferencia entre los puestos de trabajo y los servidores de Oracle.

SQLORA32.DLL. Estas dos bibliotecas las proporciona *Inprise*.

- **VENDOR INIT**: Es el nombre de la biblioteca *DLL* que proporciona el fabricante de la base de datos para conectarse con ella. Para *Oracle 8* es **OCI.DLL** y para *Oracle 7* es **ORA73.DLL.** Estas dos bibliotecas se instalan junto con el programa *Oracle 8 Client*.
- **NET PROTOCOL.** En este campo se indica el protocolo de acceso al servidor que vamos a utilizar. Generalmente será *TCP/IP* pero podemos seleccionar cualquiera de los que se muestran en la tabla 3.

No debemos confundir los conceptos índice y clave

- **OBJECT MODE.** Debemos colocarlo al valor **true** cuando trabajamos con *Oracle 8* y al valor **false** cuando trabajamos con versiones anteriores.
- **LANG DRIVER.** Aquí debemos especificar la página de códigos que utiliza el servidor *Oracle*. Si ésta no coincide con la que está utilizando nuestro *PC*, entonces tendremos problemas con los caracteres especiales como la 'ñ' o las vocales con acento.
- **SQLQRY MODE.** Debemos asegurarnos de que este parámetro tome el valor **SERVER**, pues de otra forma las consultas se ejecutarán de forma local en nuestro *PC* y no aprovecharemos la potencia que ofrece *Oracle* más que como almacén de datos.

Tenga en cuenta que todas las modificaciones que haga con este programa afectarán a todos las aplicaciones que desarrolle con *C++ Builder*.

CREACIÓN DE UNA PEQUEÑA BASE DE DATOS

Una vez configurado nuestro equipo, podemos empezar a trabajar con *Oracle*. Lo primero que haremos será crear una base de datos de ejemplo consistente en una pequeña agenda de amigos en la que guardaremos sus nombres, nacionalidades, edades, aficiones, etcétera.

EL MOTOR DE BASES DE DATOS: BORLAND DATABASE ENGINE

Borland Database Engine, BDE, es un conjunto de rutinas agrupadas en bibliotecas de enlace dinámico. *C++ Builder* usa una arquitectura en dos niveles para acceder a las bases de datos: en el nivel superior se encuentra la aplicación y en el inferior el *BDE*. El programador no ve ninguna de estas rutinas, sino que accede las bases de datos mediante los objetos definidos en la biblioteca de componentes visuales *VCL*.

El *BDE* viene preparado para manejar bases de datos locales en *dBASE*, *FOXPRO*, *MS-ACCESS* y *PARADOX*. El acceso a bases de datos remotas lo hace mediante enlaces *SQL* (*SQL Links*) y en la actualidad *Inprise* los proporciona para *DB2*, *INFORMIX*, *ORACLE*, *INTERBASE*, *SyBase* y *SQL SERVER*.

Lo más importante es que el *BDE* ofrece a las aplicaciones escritas en *C++ Builder* una interfaz de acceso a base de datos común, de tal manera que una vez preparado nuestro sistema para acceder a *Oracle* o a *SyBase*, el acceso a un servidor de bases de datos u otro desde dentro de un programa en *C++* es prácticamente igual.

CREACIÓN DE ALIAS MEDIANTE SQL EXPLORER

Junto con *C++ Builder*, al seleccionar el soporte para bases de datos, se ha instalado un programa conocido como *SQL Explorer*. Desde él se pueden manejar la mayor parte de las posibilidades que ofrece *Oracle 8* sin necesidad de aprender complicadas instrucciones o pelearnos con el obsoleto *prompt* que ofrece la herramienta *SQL*Plus* que acompaña a *Oracle*.

Crear un alias para una base de datos es tan sencillo como seleccionar la opción **Object/New** teniendo marcada la

pestaña *Database* y a continuación indicar que queremos crearla con el *driver* de *Oracle*. Le daremos el nombre de la máquina en la que resida nuestro servidor, que en nuestro caso será **murillo**, y rellenaremos la hoja de datos que aparece. Podemos dejar casi todas las opciones que aparecen a sus valores por defecto. A continuación tan sólo comentaremos el significado de algunas:

- **ENABLE INTEGERS.** Por defecto *Oracle* almacena los valores de tipo *NUMERIC* como números en coma flotante, aunque no tengan parte decimal. Al activar esta opción conseguimos que los valores *NUMERIC* cuya parte decimal sea **0** se traten como valores de tipo entero.
- **ROWSET SIZE.** Cuando trabajamos con bases de datos remotas es importante tener en cuenta que acceder a los datos es mucho más costoso que cuando trabajamos con una base de datos local. En este último caso, leer en memoria el resultado completo de una consulta para mostrarlo en una hoja de datos no suele tener mayores problemas, pero cuando los datos son remotos, tener que transferirlos por la red cuando realmente tan sólo vamos a utilizar o a mostrar en pantalla una porción de ellos puede resultar bastante costoso. Este parámetro indica cuántos registros deben transferirse entre el servidor y el *PC* o al revés en cada transferencia. Es difícil dar una regla que permita ajustar este parámetro con precisión, por lo que lo más probable es que se tengan que realizar varias pruebas hasta encontrar el valor con el que se obtiene un mejor rendimiento.
- **SERVER NAME.** El nombre o la dirección de la máquina en la que está instalado el servidor de *Oracle*. Por ejemplo **murillo.fie.us.es**.
- *USER NAME*. El nombre del usuario *Oracle* que vamos a usar para conectarnos al servidor. Podemos usar **scott** para hacer pruebas.

En el momento en que tengamos rellenos los campos podemos establecer la conexión utilizando la opción **Object/Apply**.

CREACIÓN DE TABLAS

Como inicialmente no habrá nada, lo primero que tendremos que hacer será crear la tabla de amigos colocándonos sobre la pestaña *Tables* y seleccionando la opción **Object/New**. El nombre que le asignaremos será **SCOTT.AMIGOS**. Al hacerlo se despliega una nueva lista en la que podemos añadir todas las columnas de esta tabla. En este caso, por simplicidad, tan sólo crearemos tres campos: *NOMBRE* de tipo *CHAR(30)*, *COD_NAC* de tipo *NUMBER* para guardar las códigos de nacionalidad y *COMENT* de tipo *CHAR(50)* para guardar comentarios. La tabla 4 recoge los tipos de datos que proporciona *Oracle* que se pueden utilizar desde dentro de *C++ Builder*.

En principio, *Oracle* no es sensible al uso de mayúsculas o minúsculas, por lo que da lo mismo **SCOTT.AMIGOS**, **Scott.Amigos** ó **scott.amigos** son nombres que permiten acceder a la misma tabla. Por desgracia, *SQL Explorer* tiene un pequeño problema que le impide tratar adecuadamente los nombres escritos en minúscula, por lo que deberemos evitarlos siempre. Como el campo por el que vamos a hacer referencia más a menudo es el nombre de nuestro amigo, nos colocaremos sobre la pestaña *Indices* y usaremos la opción **Object/New** para crear uno nuevo para la columna *NOMBRE*.

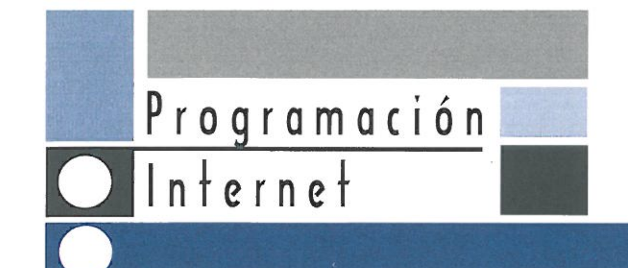
En el momento de tener los campos rellenos podremos establecer la conexión

Para definir las claves procederemos de una forma muy parecida a como lo hemos hecho con los índices. Tan sólo tenemos que seleccionar la pestaña *Primary Key* y a continuación la opción **Object/New** para elegir como clave primaria el campo *NOMBRE*. De forma parecida podemos crear una tabla llamada **SCOTT.NACIONAL** para las nacionalidades que tan sólo contendrá dos campos: *COD_NAC* de tipo *NUMERIC* y *DESCRIP* de tipo *CHAR(20)*

En el siguiente capítulo trataremos temas tan interesantes como las restricciones de integridad o las actualizaciones de datos, siguiendo para ello un ejemplo práctico.

Tabla 4. Tipos de datos en las tablas Oracle.

Tipos de datos	Significados
CHAR(1-255)	Cadena de caracteres de entre 1 y 255 caracteres
RAW(1-255)	Cadena de bytes de entre 1 y 255 caracteres
DATE	Fecha y hora actuales. Desde el 1/1/4.712AC hasta el 31/12/4.712DC
NUMBER(0-38)	Números reales con hasta 38 dígitos de precisión
LONG	Cadena de caracteres de hasta 2 gigabytes de longitud
LONG RAW	Cadena de bytes de hasta 2 gigabytes de longitud
FLOAT	Como NUMBER
VARCHAR2(1-2000)	Cadena de caracteres de longitud variable. Entre 1 y 2.000 caracteres.
VARCHAR(1-255)	Cadena de caracteres de longitud variable. Entre 1 y 255 caracteres. La diferencia con CHAR es que tan sólo se reserva espacio en la base de datos para el número de caracteres exacto que ocupa la cadena.



Creación de un buscador Web (IV)

Enrique de la Lastra (elastra@redestb.es)

Para que sea útil y práctico, un *Robot Web* debe ser capaz de analizar recursivamente todas las páginas *HTML* enlazadas a partir de una página inicial, y además debe almacenar algún tipo de información relevante de cada una de ellas. En este artículo dotaremos de esa habilidad a nuestro *Robot*.

INTRODUCCIÓN

El mes pasado implementamos un *Robot Web* que permitía conectarse a un servidor para solicitar una página *HTML* y a partir de esa página obtener una serie de variables que la definían. Estas variables servirán para almacenar una referencia, lo más completa posible, del contenido de la página y permitirán además definir unos parámetros sobre los que actuará en sucesivas ocasiones nuestro *Robot Web*.

Es decir, por un lado almacenamos qué contiene la página *HTML* y por otro lado dónde se encuentra y cuándo debemos o podemos solicitarla de nuevo. Sin embargo, tal y como dejamos el desarrollo del *Robot Web*, estaba falto de una de sus principales cualidades: el análisis recursivo de todos los enlaces de una página *HTML*. En las siguientes líneas veremos cómo añadir esta nueva funcionalidad y depuraremos un poco el código que ya estaba escrito, en previsión de futuras necesidades.

LIMITACIONES

Cómo no, el *Robot* va a adolecer de algunas limitaciones. La primera consiste en que, todavía, no almacena las páginas visitadas en una base de datos. Por ahora sólo se guarda la información filtrada en una lista en memoria. Para cada enlace analizado se guarda una estructura de datos (*TPaginaHTML*), que previamente hemos rellenado utilizando el propio contenido de la página *HTML*.

Aunque resulte reiterativo, recordamos que el *Robot* sólo indexará páginas *HTML*, y no otro tipo de recursos *web* (imágenes, sonidos, ficheros de vídeo, direcciones de correo electrónico, etc.). Tampoco recuperará (ni por tanto hará caso) del contenido del fichero **ROBOTS.TXT**, y por tanto se seguirá comportando de forma políticamente incorrecta.

> Guardamos la información de cada página HTML en una estructura de datos y ésta a su vez en una lista en memoria

Y por último y no menos importante, la principal limitación que tiene el *Robot* es que no comprueba si una conexión con un servidor *web* no se puede establecer, ni permite ignorar un

enlace (y pasar al siguiente) y en general se despreocupa de los posibles errores ocurridos en cada conexión.

NUEVAS VARIABLES

En el artículo del mes pasado definimos la estructura de datos que íbamos a asociar a cada página *HMTL*. Como queremos almacenar esta información en una lista en memoria, y *Delphi* suministra la clase *TList* para el manejo de listas de objetos, habrá que definir un puntero a esa estructura de datos.

Además declararemos dentro de la clase del *Robot* una variable de tipo *TList* que será la que almacene la lista de páginas visitadas (o pendientes de ser analizadas):

```
pTPaginaHTML = ^TPaginaHTML;
TPaginaHTML = record
  EstadoPagina: TEstadoPaginaHTML;
  Referer: string;
...
end;

type
 TRobotForm = class(TForm)
 ...
private
 { Lista con la información de cada página analizada por el Robot }
  ListURL: TList;

 { Información acerca de la página HTML,
     incluidos los datos de la propia petición }
  PaginaHTML: pTPaginaHTML;
 { Variable auxiliar para la asignación dinámica
     de memoria }
  PagHTML: pTPaginaHTML;
 ...
```

Para el manejo de la lista, definimos unas variables en la estructura *TPaginaHTML* que permitirán conocer para cada página analizada, cuál es la posición en la lista del primero de sus enlaces, cuál es el enlace que en un instante dado está siendo analizado y cuál es la posición en la lista del **enlace padre** (es decir, el enlace que nos trajo a la página en cuestión).

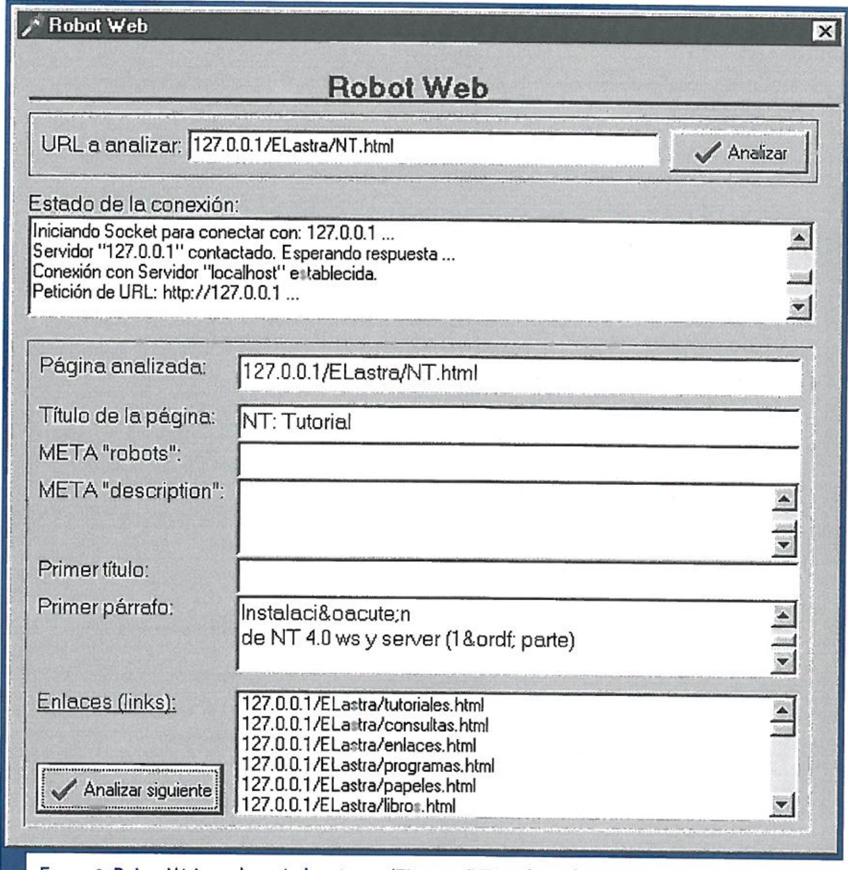


Figura 1. Robot Web analizando la página /ELastra/NT.html en el Host Local empleando como servidor el Personal Web Server.

El Robot sólo indexará páginas HTML, y no otro tipo de recursos web

El orden de almacenamiento en la lista es el siguiente:

1. Se almacena la página introducida por el usuario.
2. Se crean posiciones contiguas en la lista para todos y cada uno de los enlaces de la página inicial.
3. Cada vez que se analiza una página y contiene enlaces, se crean nuevas posiciones contiguas (a partir de la última) para cada uno de los enlaces de esa página.
4. Y así hasta llegar al último enlace de la página final.

Como vamos a manejar una lista, y teniendo en cuenta que esa lista contiene los enlaces de cada página *HTML*, necesitaremos un par de variables para conocer dónde nos encontramos en la lista en cada instante y cuál es el **enlace padre** de la página *HTML* que está siendo analizada:

```
private
 { Índice de la página que está siendo analizada
     e índice de la página "padre" cuyos
     enlaces están siendo analizados }
  PaginaAnalizada: integer;
  PaginaPadreAnalizada: integer;
```

Una variable adicional que vamos a utilizar para definir el comporta-

Listado 1. Modificaciones realizadas en el procedimiento: "ProcesarPaginaHTML".

```
function TRobotForm.ProcesarPaginaHTML (Socket: TCustomWinSocket;
Recibido: string): boolean;
var
 EtiquetaRobots: string;
 i: integer;
begin
...
...
{ Si se puede analizar la página, la analizamos }
 if PaginaHTML^.Follow then begin
  BuscarEnlaces (Recibido, PaginaHTML^.Enlaces);
  if Assigned(PaginaHTML^.Enlaces) and (PaginaHTML^.Enlaces.Count > 0) then
  begin
   ListBoxEnlaces.Items.Clear;
   ListBoxEnlaces.Items.AddStrings (PaginaHTML^.Enlaces);

   pTPaginaHTML(ListURL.Items[PaginaAnalizada])^.EnlacePrimero :=
       InicializarPaginaHMTL (PaginaAnalizada);

   for i:= 1 to PaginaHTML^.Enlaces.Count - 1 do begin
    InicializarPaginaHMTL (PaginaAnalizada);
   end;
  end
 end;
 { Si la página analizada es la última de la página padre, actualizamos el
  número de índice de la página padre analizada }
   if Assigned(pTPaginaHTML(ListURL.Items[PaginaPadreAnalizada])^.Enlaces)
   and (pTPaginaHTML(ListURL.Items[PaginaPadreAnalizada])^.Enlaces.Count > 0)
   then begin
  if (pTPaginaHTML(ListURL.Items[PaginaPadreAnalizada])^.EnlacePrimero +
    (pTPaginaHTML(ListURL.Items[PaginaPadreAnalizada])^.Enlaces.Count - 1)) =
    PaginaAnalizada
  then begin
    i := pTPaginaHTML(ListURL.Items[PaginaPadreAnalizada])^.EnlacePrimero;
    while pTPaginaHTML(ListURL.Items[i])^.Enlaces.Count = 0 do Inc(i);
    PaginaPadreAnalizada := i;
  end;
 end;

 if FuncionamientoRobot = frManual then
  BtnSiguiente.Enabled := True
 else if FuncionamientoRobot = frAuto then
  AnalizarSiguiente;
End;
```

miento del *Robot* es *Funcionamiento-Robot*, también esta misma variable servirá para poder determinar el modo de trabajo del motor: **manual** o **automático**.

En el modo **manual**, que será útil para depurar y observar el funcionamiento del *Robot*, tendremos que pulsar el botón **AnalizarSiguiente** cada vez que queramos analizar el siguiente enlace en la lista. En el modo **automático** este análisis arrancará de forma completamente automática y finalizará cuando lleguemos al último enlace de la lista (si es que es posible) o más probablemente, cuando nos quedemos sin memoria o se genere un error.

El Robot tendrá dos modos de trabajo: manual y automático

Como sólo puede almacenar dos posibles valores declararemos un tipo enumerado y una variable de este tipo:

```
Type
 TFuncionamientoRobot = (frManual, frAuto);

TRobotForm = class(TForm)
...
private
 { Tipo de funcionamiento del Robot: manual o
     automático }
 FuncionamientoRobot: TFuncionamientoRobot;
```

El modo de trabajo por defecto será el **manual**, que permitirá observar por pantalla las evoluciones del *Robot Web*. En cuanto a la lista de enlaces, cada vez que el usuario introduzca en el cuadro de texto del *Robot URL a analizar* un nuevo *URL* y pulse el botón **Analizar**, destruiremos la lista y crearemos una nueva (y vacía). Para destruir la lista, crearemos el procedimiento *DestruirListaURLs*, al que llamaremos al destruir el formulario y cuando el usuario introduzca un nuevo *URL* para analizar.

ANÁLISIS DE URLS

Al pulsar el botón **Analizar** crearemos la lista de *URLs* (después de destruir la anterior, ya que pueden haber sido analizados antes otros), e inicializaremos

las variables asociadas a la lista. En la variable *PaginaHTML* (que ahora es un puntero a una estructura de datos de tipo *TPaginaHTML*), almacenaremos la dirección de la primera posición que figura en la lista.

En cuanto al procedimiento *SepararHost_y_URI*, ha habido que retocarlo para adaptarlo a la variable de tipo puntero que utilizamos. La razón es que *Delphi* no permite pasar parámetros por variable cuando estos provienen de un puntero a una estructura de datos. Por tanto, hay que definir unas variables auxiliares que recojan la salida del procedimiento *SepararHost_ y_URI*. Esta variación está en el código del CD-ROM.

Pulsando el botón Analizar creamos la lista de URLs

El cuerpo del manejador del evento *OnClick* del botón **Analizar Siguiente** sigue la misma estructura que el del botón **Analizar**, pero con unas ligeras variaciones. La primera es que cada vez que entramos en el procedimiento incrementamos la variable **PaginaAnalizada**, que indica la posición en la lista de la página que vamos a analizar. La segunda variación, es que ya no utilizamos el cuadro de texto *URL a analizar* para obtener el *URL*, sino que lo obtenemos directamente de la posición donde previamente lo habremos almacenado en la lista.

Por último y muy importante, abriremos una nueva conexión del *socket* sólo si el servidor *Web* al que se dirige la petición *HTTP* de la siguiente página es diferente al de la página *HTML* anteriormente solicitada. La razón para operar de esta manera es que el protocolo *HTTP/1.1* establece que las conexiones, por defecto, se deben mantener abiertas, con el fin de ahorrar recursos de apertura y cierre de *sockets*. El código completo se puede observar en el Listado que está en el CD-ROM de la revista.

Figura 2. Robot Web analizando uno de los enlaces contenidos en la URL introducida por el usuario (ver Figura 1).

OTRAS MODIFICACIONES

Los manejadores de los eventos: *OnLookup*, *OnConnecting*, *OnConnect* y *OnRead*, permanecen inalterados. De igual manera, permanecen sin variación el procedimiento *ProcesarCabecerasRespuesta* y todos los procedimientos de análisis de etiquetas.

El procedimiento que sí cambia es *ProcesarPaginaHMTL*, ya que ahora tiene que almacenar en la lista tantas posiciones como enlaces haya contenidos en cada página, y lo que es más importante, mantener actualizadas las variables **PaginaAnalizada** y **PaginaPadreAnalizada** que permiten conocer la posición, dentro de la lista, de la página que está siendo analizada y de la página con un enlace hacia aquélla.

Además aquí donde activaremos el modo **automático** de rastreo del *Robot*. El código se observa en el Listado 2, donde se ha incluido sólo el código modificado respecto a la versión anterior .

CONCLUSIÓN

Hemos visto cómo añadir una de las funcionalidades más importantes de nuestro *Robot Web*: la posibilidad de analizar recursivamente todos los enlaces *HTML* a partir de una *URL*. Hemos desarrollado el código necesario para manejar la lista de enlaces y almacenar la información relevante sobre cada página. En la próxima entrega, depuraremos los posibles errores de conexión del *Robot* y emplearemos una base de datos para almacenar la información que previamente hemos clasificado en una lista.

Dudas técnicas

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en algún tema relacionado con la programación en cualquier entorno.

Pregunta

Ante todo me gustaría felicitaros por la revista tan estupenda que hacéis y quiero animaros a que sigáis en esta línea. Ahora quisiera plantearos algunas de las varias cuestiones que tengo sobre LINUX.

1.- En las transferencias con FTP, parece que la conexión muere cuando hago un put, sin embargo si hago get funciona perfectamente. ¿Qué ocurre?

2.- ¿Cómo debo usar el control de flujo XON/XOFF?

3.- ¿El módem parece que conecta a velocidades extrañas? Cuando uso minicom, el módem siempre usa 14400 bits/segundo. Sin embargo, PPP dice que está conectando a 9600, 7200 e incluso a 2400 bits/segundo. ¿Cómo puedo corregir esto?

4.- Cuando hago FTP, la operación get es muy lenta, pero la operación put, sin embargo, es muy rápida. ¿Porqué?

5.- La opción proxyarp no encuentra la dirección hardware.

Respuesta

Estimado lector vamos a intentar resolver tus dudas una por una:

1.- ¿Está activado el control de flujo (*flow control*)?. Esto se hace pasando a *pppd* la opción **crtscts** para usar control de flujo *RTS/CTS* (hardware) o *XON/XOFF* para control de flujo *XON/XOFF* (software). Si no tienes habilitado el control de flujo, probablemente está sobreescribiendo en los buffers del módem. Esto tiene consecuencias catastróficas si utiliza compresión de cabeceras *vj* (*Van Jacobson*).

2.- Es mejor utilizar control de flujo hardware (*CTS/RTS*). Sin embargo, si te ves obligado a usar control de flujo software, haz lo siguiente:

- Necesitas especificar la opción *XON/XOFF* en la línea de comandos de *pppd*. Esta opción le dice al dispositivo serial a utilizar que utilice este tipo de control de flujo. Además, carga los dos caracteres (*XON y XOFF*) dentro del *driver tty*.
- Necesitas especificar los caracteres que representan *XON* y *XOFF* en el parámetro *asyncmap* que se pasa a *pppd*. Esto avisa al sistema remoto que debe separar estos caracteres cuando quiera enviárselos a su máquina. Esto se indica normalmente con la opción **asyncmap** a0000.
- Naturalmente, no olvides decirle a tu módem que utilice control de flujo *XON/XOFF*. En los módem *ZyXEL*, se suele utilizar la secuencia **"R1&H4"**.

3.- Especifica la velocidad que deseas en la línea de comandos de *pppd*. Si no especificas la velocidad, *PPP* utilizará cualquier velocidad que exista. Algunos programas no dejan los parámetros de la línea serial iguales que cuando se ejecutaron. Esto puede causar que la línea tenga una configuración extraña.

Linux no soporta modems que utilizan *RPI* (*Rockwell Protocol Interface*) porque es un protocolo propietario. Dado que *Rockwell* no quiere facilitar el código necesario para poder hacer una adaptación a *Linux*, hay muy pocas posibilidades de que estos módem sean soportados por *Linux*. La solución en este caso es clara: no usar modems *RPI*.

Si no sabes si un módem es RPI cuando quieras adquirirlo, fíjate en las frases publicitarias que aparecen en la caja. Frases del estilo "con corrección de errores software", o "compatible con *Windows*" o "requiere un *driver* especial para funcionamiento completo", usualmente suelen indicar que el módem es *RPI*.

4.- ¿Especificaste la opción *asyncmap* 0 cuando ejecutó *pppd*?. Si lo olvidaste, el peer debe doblar todos los caracteres de control en el rango **0x00..0x1F** (hexadecimal). Esto supone una reducción de velocidad de un 12.5 % cuando está recibiendo datos.

5.- Usa el paquete **ppp-2.1.2d.tar.gz.** El proceso **pppd** fue compilado erróneamente con el *kernel 1.1.8* y usaba definiciones *Net-3* en vez de la *Net-2* como le correspondía.

El paquete 2.1 tiene establecido un límite de 64 dispositivos de red. Cuando se escribió el código de *proxyarp* se pensó que era un número razonable, dado que la mayoría de la gente suele tener uno o dos controladores *Ethernet* como máximo en una máquina. Hoy en día hay máquinas que tienen conectados hasta 128 dispositivos de red.

La versión 2.2 ha elevado el límite hasta 256 dispositivos de red. Este límite aparece reflejado en forma de un # y define que se encuentra en el módulo *sys-linux.c*.

Pregunta

Lo primero que hago es felicitaros por uno de los cursos que más me han gustado de la revista, se trata del de DHTML que publicáis desde hace algunos números y que pretendo seguir al completo ya que me parece muy bien explicado y muy útil, pues en ningún otro sitio se han atrevido con el DHTML COMPATIBLE.

Mi problema es el siguiente. Se dice en el primera entrega del curso que se puede acceder a todos los elementos de la página HTML. Pues bien, no encuentro la manera de meter en una capa los elementos de los formularios. En el Explorer sí funciona, pero con Netscape no. He usado las funciones de la primera entrega del curso, y funciona perfectamente con imágenes, por ejemplo, pero no con formularios. ¿Es que Netscape no soporta capas para los formularios? Muchas gracias de antemano, y por favor responderme.

Respuesta

Estimado lector antes de nada queremos darte las gracias por leer tan atentamente los artículos de la revista. Sinceramente creemos que no deberías tener problemas para meter un formulario dentro de una capa y manejar la capa. Nosotros lo hemos hecho varias veces y funciona tanto en *IE4* como en *NS4*. Trataremos de solucionarte el problema de un modo sencillo.

Por ejemplo:

```
<DIV ID="micapa" CLASS="miclase">
<FORM NAME="miform" ACTION="/cgi-bin/micgi.cgi" METHOD="POST"><INPUT
TYPE="Text" NAME="NOMBRE"><INPUT
TYPE="Text"
NAME="APELLIDO"></FORM></DIV>
```

Esa capa la puedes manejar tal y como haces con cualquier otra (ocultarla, moverla, etc.). No hay ninguna diferencia. El único problema que puedes tener es si quieres acceder desde un *script* al formulario aunque no es realmente un problema, se trata simplemente de otra diferencia entre *NS4* y *IE4* que hay que salvar. Para *IE4* todos los formularios son accesibles desde el objeto documento principal, estén o no estén en una capa. Para *NS4* un formulario que se encuentre en una capa pertenece al objeto documento de la capa.

Este problemilla lo puedes evitar creándote una referencia universal al formulario.

Así tienes que:

```
var f_miform; function iniciar() { ... c_micapa =
new objetoCapa('micapa'); ... if (NS4) { f_miform
= c_micapa.document.forms["miform"]; } else
{ f_miform = document.forms["miform"]; } ... }
```

De esta forma y a partir de ese momento vamos a poder referirnos en este *script* a cualquier campo del formulario.

Por ejemplo: alert(f_miform.APELLIDO.value);y funcionará tanto en IE4 como en NS4. Esperamos haber resuelto tu problema, y te animamos a que sigas consultándonos las dudas que tengas o necesites aclarar.

Pregunta

Soy una aficionada lectora de su revista desde hace algún tiempo y, recientemente, han llegado a mis manos algunos ejemplares atrasados. En concreto se trata de un artículo sobre Sistemas Distribuidos.

He encontrado el artículo muy interesante, por lo que voy a intentar adquirir el resto de las revistas que componen la serie.

Los ejemplos de código que se incluyen en el CD-ROM son también muy interesantes, pero a pesar de que domino el lenguaje de programación C, encuentro algunos problemas al seguir el flujo del código que incluyen con los artículos.

En concreto mis principales dificultades se deben a una serie de líneas similares a esta que os comento a continuación:

```
TimeKey = ( OldKey >= NewKey ) ? (OldKey
    | NewKey ) : (NewKey & OldKey);
```

En mi experiencia con Visual C++ y otros compiladores, nunca me he encontrado con algo de este estilo, por lo que me pregunto si será parte del lenguaje C, o algo muy particular de Linux.

Si me pudieran ayudar a desentrañar este fragmento de código, les estaría muy agradecidos, ya que encuentro que los Sistemas Distribuidos son un tema novedoso e interesante.

Muchas gracias por adelantado

Respuesta

En el álgebra del lenguaje *C*, existen un gran número de operadores. Todos ellos pueden ser utilizados por el programador para manipular los diferentes datos del programa, así como para controlar el flujo del mismo.

La mayoría de ellos son operadores binarios, requiriendo dos datos para poder ser utilizados. Los ejemplos más clásicos son los siguientes:

Comparación: se utilizan para evaluar el valor de verdad de una condición. Por ejemplo podemos encontrarnos con:

```
a > 3,  c != b, d == ( 3 * 17 ) , etc...
```

Asignación: se utilizan para modificar el valor de una variable en el curso de un programa. Como ejemplos vemos los siguientes:

```
c = 45,  Key *= 22, Logo += 13, etc...
```

Aritméticos: utilizados generalmente para realizar operaciones sobre un grupo de datos.

```
45 + 17, 75 / 12, 89 * 34, etc...
```

Sin embargo, existen muchos más operadores en *C*, utilizados para diversas funciones. Entre ellos destacan los operadores lógicos, y el operador *'?'*. Los operadores lógicos más utilizados son los siguientes:

Operador *OR (| |)*: permite establecer disyunciones lógicas entre dos operandos. La siguiente expresión se podría leer como: "Si *'a'* es igual a tres o *'a'* es igual a cinco".

```
If( ( a == 3 ) | | ( a == 5 ) )
```

Operador *AND (&&)*: de manera especular al anterior, permite utilizar la conjunción lógica entre dos expresiones. La expresión que vemos a continuación significa: "Si *'a'* es mayor que tres y *'a'* es menor que cinco".

```
If( ( a > 3 ) && ( a < 5 ) )
```

Operador *OR* de bit *(|)*: este operador permite realizar un *OR* lógico, bit a bit, entre dos operandos:

```
c = a | b; ( c es igual al OR de bit de 'a' y 'b' )
```

Suponiendo que *'a'* , *'b'* y *'c'* son datos de 8 bits *(char)*, y sus valores son los siguientes:

```
a = 10001000
b = 00010001
```

El resultado quedará:

```
c = ( 10001000 OR 00010001 ) = 10011001
c = 10011001
```

Operador *AND* de bit *(&)*: caso similar al anterior, pero con la conjunción del álgebra de Boole aplicada al conjunto de los números binarios:

```
c = a & b ( c es igual al AND de bit de 'a' y 'b' )
```

Esta vez los tres datos son de 16 bits, - *short*, - y sus valores son:

```
a = 1111000011110000
b = 1100110011001100
```

El resultado será el que mostramos a continuación:

```
c = 1111000011110000 AND
        1100110011001100 = 1100000011000000
c = 1100000011000000
```

Finalmente, vamos a analizar el operador *'?'*, su sintaxis es un tanto peculiar, ya que incluye tres operandos, separados por dos símbolos: *'?'* y *':'*. Cualquier expresión de este estilo está organizada de la siguiente manera:

```
Expresión_1 ? Expresión_2 : Expresión_3 ;
```

Su significado se obtiene:

1.- Se analiza **Expresión_1**, comprobando si su valor es igual a cero (falso) o no (cierto).

2.- Si la expresión, - típicamente una condición, - es cierta, toda la línea será igual a **Expresión_2.**

3.- Si es falsa, entonces el resultado será **Expresión_3.**

Utilizando un ejemplo para clarificar el comportamiento, tenemos:

```
( a < b ) ? a : b
```

Esta expresión da como resultado *'a'* si ésta es menor que *'b'*. En caso contrario, obtenemos *'b'*. De esta manera, la sentencia *C* siguiente:

```
if ( a < b )
z = a;
else
z = b;
```

Se puede reescribir como:

```
z = ( a < b ? a : b );
```

Por lo tanto, la línea de código que nos envías tiene la siguiente interpretación:

"Si *OldKey* es mayor o igual que *NewKey*, entonces *TimeKey* vale el *OR* binario de *OldKey* y *NewKey*. En caso contrario, *TimeKey* es igual al *AND* binario de *NewKey* y *OldKey*."



☐ **Sí, deseo suscribirme a la revista SÓLO PROGRAMADORES durante un año (12 números) eligiendo la siguiente modalidad.** (Marcar con una X)

☐ Suscripción **NORMAL**
9.500 ptas. / 57,10 €

☐ Suscripción **ESTUDIANTES**
7.600 ptas. / 45,68 €

Nombre ....Apellidos ....

Domicilio: calle ....

C.P: ....Población:...Nº..............................Piso ....

Teléfono: ....Provincia: ....

....Fax: ....E-mail: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº: .... Fecha de caducidad ..../.........

☐ Domiciliación bancaria. (Rellenar código cuenta cliente)

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS, S.R.L. que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

• ESTA OFERTA ANULA LAS ANTERIORES

• Oferta válida hasta fin de existencias • Válido para Península y Baleares • Canarias consultar

CÓDIGO DE CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

SÓLO PROGRAMADORES

☑ **Sí, deseo recibir los siguientes números atrasados** ....

....

.... **de SÓLO PROGRAMADORES al precio unitario de 995 ptas. (IVA incluido)**

números 1, 2, 3, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 23, 24 y 26 agotados

Nombre ....Apellidos ....

Domicilio: calle ....

C.P: ....Población:...Nº..............................Piso ....

Teléfono: ....Provincia: ....

....Fax: ....E-mail: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº: .... Fecha de caducidad ..../.........

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS, S.R.L. que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

• ESTA OFERTA ANULA LAS ANTERIORES

• Oferta válida hasta fin de existencias • Válido para Península y Baleares • Canarias consultar

Firma:

PEGAR AQUÍ

Rte.: ....

....

....

RESPUESTA COMERCIAL
Autorización nº 14.107
B.O. de C. nº 36
del 18.04.97

NO
NECESITA
SELLO
(A franquear
en destino)

HOJA DE PEDIDO



APARTADO FD Nº 214
28080 MADRID